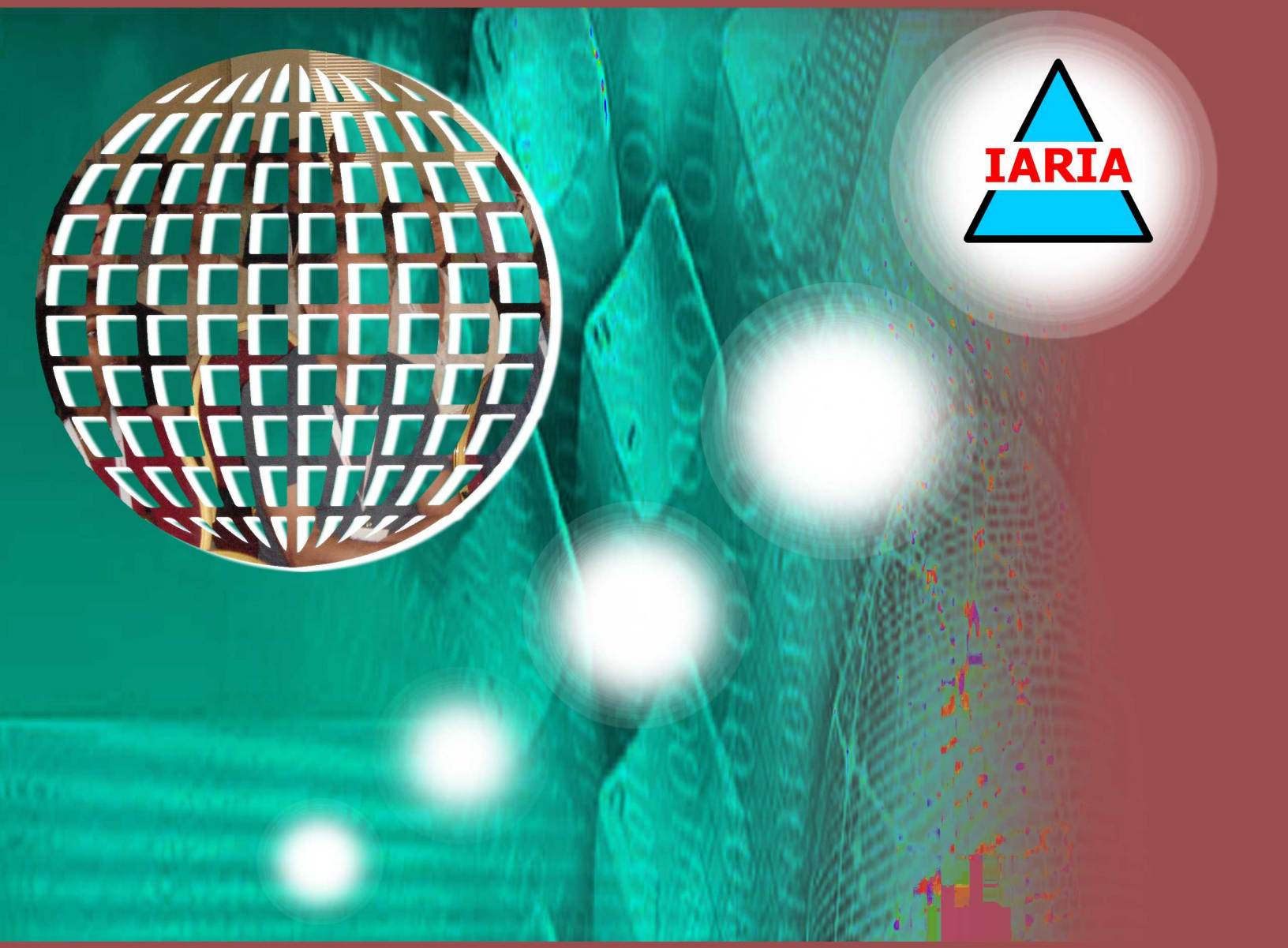
International Journal on Advances in Software



2009 vol. 2 nr. 1

The *International Journal On Advances in Software* is Published by IARIA.
ISSN: 1942-2628
journals site: http://www.iariajournals.org
contact: petre@iaria.org

Sponsored by IARIA
www.iaria.org

International Journal On Advances in Software
Volume 1, Number 1, 2008

- Constantin Paleologu, University 'Politehnica' of Bucharest, Romania
- Liviu Panait, Google Inc., USA
- Kenji Saito, Keio University, Japan
- Ashok Sharma, Satyam Computer Services Ltd – Hyderabad, India
- Marcin Solarski, IBM-Software Labs, Germany

Advanced Computing

- Matthieu Geist, Supelec / ArcelorMittal, France
- Jameleddine Hassine, Cisco Systems, Inc., Canada
- Sascha Opletal, Universitat Stuttgart, Germany
- Flavio Oquendo, European University of Brittany - UBS/VALORIA, France
- Meikel Poess, Oracle, USA
- Kurt Rohloff, BBN Technologies, USA
- Said Tazi, LAAS-CNRS, Universite de Toulouse / Universite Toulouse1, France
- Simon Tsang, Telcordia Technologies, Inc. - Piscataway, USA

Geographic Information Systems

- Christophe Claramunt, Naval Academy Research Institute, France
- Dumitru Roman, Semantic Technology Institute Innsbruck, Austria
- Emmanuel Stefanakis, Harokopio University, Greece

Databases and Data

- Peter Baumann, Jacobs University Bremen / Rasdaman GmbH Bremen, Germany
- Qiming Chen, HP Labs – Palo Alto, USA
- Ela Hunt, University of Strathclyde - Glasgow, UK
- Claudia Roncancio INPG / ENSIMAG - Grenoble, France

Intensive Applications

- Fernando Boronat, Integrated Management Coastal Research Institute, Spain
- Chih-Cheng Hung, Southern Polytechnic State University, USA
- Jianhua Ma, Hosei University, Japan
- Milena Radenkovic, University of Nottingham, UK
- DJamel H. Sadok, Universidade Federal de Pernambuco, Brazil
- Marius Slavescu, IBM Toronto Lab, Canada
- Cristian Ungureanu, NEC Labs America - Princeton, USA

Testing and Validation

- Michael Browne, IBM, USA
- Cecilia Metra, DEIS-ARCES-University of Bologna, Italy
- Krzysztof Rogoz, Motorola, USA
- Sergio Soares, Federal University of Pernambuco, Brazil
- Alin Stefanescu, SAP Research, Germany

- Massimo Tivoli, Universita degli Studi dell'Aquila, Italy

Simulations

- Robert de Souza, The Logistics Institute - Asia Pacific, Singapore
- Ann Dunkin, Hewlett-Packard, USA
- Tejas R. Gandhi, Virtua Health-Marlton, USA
- Lars Moench, University of Hagen, Germany
- Michael J. North, Argonne National Laboratory, USA
- Michal Pioro, Warsaw University of Technology, Poland and Lund University, Sweden
- Edward Williams, PMC-Dearborn, USA

International Journal On Advances in Software
Volume 1, Number 1, 2008

Foreword

The first 2009 number of the International Journal On Advances in Software compiles a set of papers with major enhancements based on previously awarded publications. It brings together a set of articles that share a common link to software. For this issue, thirteen contributions have been selected.

In the first article, Javier Díaz et al. present a heuristic approach to allocating workloads in computational grid environments. Experiments were done with several workload distributions. It was found that a Simulates Annealing approach significantly reduces computation time.

The second article by Torsten Hopp et al. present a Matlab GUI that can facilitate dealing with the large amount of data that is generated by ultrasound tomography. The proposed approach has a smaller learning curve and is extensible via plugins.

Sascha Opletal et al. consider aspects of design information. The proposal is an automated setup and classification of information pieces within several involved knowledge domains.

The fourth article by Kaouthar Fakhfakh et al. address the shortcomings of current SLA models in service-oriented architectures. Through the use of SWRL (Semantic Web Rule Language), a set of SLA obligations can be defined as well as actions to handle violations of the agreements.

Marc Aiguier et al. address the issue of formalization needed for large software systems. A mathematical denotation is proposed for the notion of complex software systems whose behavior is specified by rigorous formalisms. A denotation is built on Goguen's institution theory.

Heiko Pfeffer introduces a service composition model that can underlie modern Web applications and can be executed within a respective runtime at every browser-enabled client. The resulting Underlay System for Web Mashups provides uniformity irrespective of client technology.

In the next article, S. E. Hegazy and C. D. Buckingham use the case of clinical decision support systems to present a knowledge based structure for reasoning with uncertainty. Such an approach can be generalized to similar knowledge-engineering domains where relative weightings of node siblings are part of the parameter space.

Joerg Bartholdt et al. address variability in the data models and how this is reflected in software. The challenge is to have proper adapters generated in order to assist integration. An eHealth case study is presented which shows that information can be consistently preserved while maintaining the desirable aspects of the software.

A Kernel-based Bayesian Filtering Framework is proposed by Matthieu Geist at al. in order to support learning at the level of function calls. By quantifying uncertainty, the presented solution can deal with cases where actual samples are not directly observable.

Joey C. Libby et al. tackles the issue of computationally intensive problems. A hardware approach was taken in this case, which with additional optimizations yields promising results.

Unit test generation is the focus of the article by Alberto Bacchelli et al. The contribution presents a practical comparison between different automated methods, tools, and techniques to generate unit tests. A testing procedure based on bet practices is presented.

In the spirit of consistency and correctness in large scale workflows, Osamu Takaki et al. propose an incremental verification approach. In order to verify a workflow incrementally, it is necessary to consider consistency properties of not only a whole workflow but also a subgraph of the whole workflow. The work presented defines a consistency property for evidence life cycles in workflows with multiple starts.

Finally, Jaroslav Kral and Michal Zemlicka discuss a set of service-oriented antipatterns and their impact on software engineering. The development risks associated with antipatterns are presented as they can affect entities from small firms to government sized applications.

We hope that the contents of this journal will add to your understanding of software, and that you will be inspired to contribute to IARIA's conferences that include topics relevant to this journal.

Jon G. Hall, Editor-in-Chief
Petre Dini, IARIA Advisory Committees Board Chair

International Journal On Advances in Software
Volume 2, Number 1, 2009

CONTENTS

A Heuristic Approach to the Allocation of Different Workloads in Computational Grid Environments

Javier Díaz, Sebastián Reyes, Camelia Muñoz-Caro and Alfonso Niño
Grupo de Química Computacional y Computación de Alto Rendimiento,
Escuela Superior de Informática, Universidad de Castilla-La Mancha,
Paseo de la Universidad 4, 13071, Ciudad Real, Spain
E-mail:{javier.diaz,sebastian.reyes,camelia.munoz,alfonso.nino}@uclm.es

Abstract

Self-scheduling algorithms can achieve a good balance between workload and communication overhead in computational systems. In particular, Quadratic Self-Scheduling (QSS) and Exponential Self-Scheduling (ESS) are flexible enough to adapt to distributed systems. Thus, they are of interest for application in Internet-based Grids of computers. We have tackled the problem of scheduling a set of independent tasks in a computational Grid using a simulator and a heuristic approach based in simulated annealing. To test this approach, we have considered different workload distributions. We find that the optimal Simulated Annealing (SA) results permit to reduce the overall computing time of a set of tasks up to a 16%, with respect to results obtained with previous values of the parameters experimentally determined. In addition, the time to obtain the SA optimized parameters by simulation is negligible compared with that needed using experimental measures. Moreover, after the optimization, the heuristic approach provides equivalent performance for different workloads (random, increasing, decreasing) and different number of tasks. These results show the high adaptability of the QSS and ESS self-scheduling algorithms, which can be fully exploited thanks to the heuristic approach here presented.

Keyword: Self-Scheduling Algorithms, Heuristic Scheduling, Computational Grid.

1. Introduction

A computational Grid [14] is a hardware and software infrastructure providing dependable, consistent, and pervasive access to resources among different administrative domains. The objective is to enable the sharing of these resources in a unified way, maximizing their use. A Grid can be used effectively to support large-scale runs of distributed applications. An ideal case to be run in Grid is that with many large independent tasks. This case arises naturally in parameter sweep problems. A correct assignment of tasks, so that computer loads and communication overheads are well balanced, is the way to minimize the overall computing time. This problem belongs to the active research topic of the development and analysis of scheduling algorithms. Different scheduling strategies have been developed along the years (for the classical taxonomy see [6]). In particular, dynamic self-scheduling algorithms are extensively used in practical applications. These algorithms represent adaptive schemes where tasks are allocated in run-time. Self-scheduling algorithms were initially developed to solve parallel loop scheduling problems in homogeneous memory-shared systems, see for instance [24]. Self-scheduling algorithms divides the set of tasks into subsets (chunks), and allocates them among the processors. In this way overheads are reduced. However, the performance of a self-scheduling algorithm is not independent of the workload distribution. The workload represents the tasks duration distribution in a problem. Figure 1 shows the four possible cases: uniform workload, increasing workload, decreasing workload, and random workload. They have a direct influence in the performance of scheduling algorithms. In general, the random case is the most difficult to schedule, since the duration of the different tasks is not known beforehand. However, the increasing workload can be optimally scheduled using a decreasing chunk distribution function. This is because it assigns a large number of small tasks at first, and a few number of big tasks at the end, trying to guarantee a good load balance. On the other hand, the decreasing workload can be scheduled efficiently using an increasing chunk distribution function.

Self-scheduling algorithms have been tested successfully in distributed memory multiprocessor systems and heterogeneous clusters [3][7][17][22][27][31]. In addition, some works about their performance on Grid connected clusters

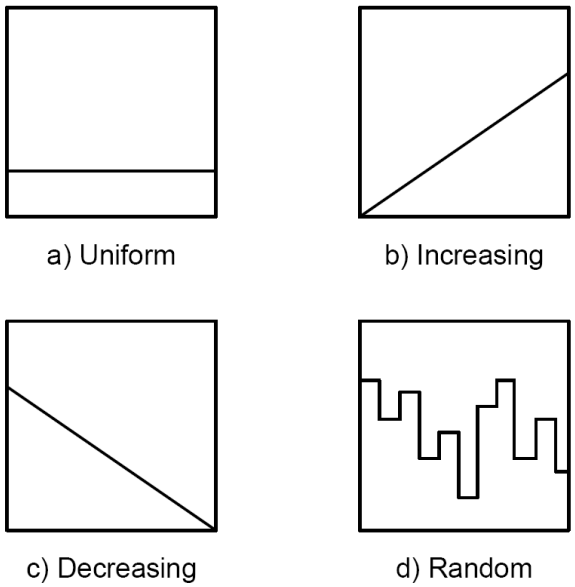


Figure 1. The four different workloads. a) uniform workload, b) increasing workload, c) decreasing workload, d) random workload

of computers have been reported [8][9][26][29]. Thus, although self-scheduling algorithms are derived for homogeneous systems, in principle, they can be applied to heterogeneous ones, such as computational Grids [14]. However, they could be not enough flexible (they may have not enough degrees of freedom) to adapt efficiently to a heterogeneous environment. In this sense, we have previously proposed two new flexible self-scheduling algorithms called Quadratic Self-Scheduling (QSS) [7][8] and Exponential Self-Scheduling (ESS) [9][10]. The first is based on a quadratic form for the chunks distribution function. Therefore, it has three degrees of freedom, which provide high adaptability to distributed heterogeneous systems. The second approach, ESS, is based on the slope of the chunks distribution function. In this case, we consider that the chunks distribution function decreases in an exponential way. This algorithm provides a good adaptability in distributed heterogeneous systems through two parameters. Moreover, in previous works [9][10] we have compared our approaches with other self-scheduling algorithms in an actual Grid environment. The new algorithms outperform the previous ones, since they obtain better load balance and more reduction of the communication overhead [9].

However, a computational Grid is made up of a large number of independent resource providers and consumers, which are running concurrently, changing dynamically, and interacting with each other. Due to these environment characteristics, new approaches such as those based in heuristic algorithms [20][11] have been proposed to address the challenges of Grid computing. These kinds of algorithms make realistic assumptions based on a priori knowledge of the concerning processes and of the system load characteristics. Braun et al [4] presented three basic heuristics, based on Nature, for Grid scheduling. These are Genetic Algorithms (GA), Simulated Annealing (SA) and Tabu Search (TS).

Genetic algorithms (GA) [16] provide a robust searching technique that allows a high-quality solution to be derived from a large search space. This solution is obtained in polynomial time by applying the principle of evolution. One of the different uses of these algorithms is in Grid Scheduling as seen in [2][23][30]. A genetic algorithm combines exploitation of best solutions from past searches with the exploration of new regions of the solution space. A genetic algorithm for scheduling problems can be organized as follows [19]. First, an initial population is necessary, which can be generated by other heuristic algorithm. A population is a set of “chromosomes” where each represents a possible solution. A solution is a mapping sequence between tasks and machines. The “chromosomes” are evaluated and a fitness value is associated with each. The fitness value indicates how well the individual is compared with others in the population. Next, the population evolves, that is, a new generation is obtained by using the genetic operators, namely selection, crossover and mutation [16]. Finally, the chromosomes from this modified population are evaluated again. This completes one iteration of the GA. The GA stops when a predefined number of iterations are reached or all chromosomes converge to the same mapping.

Another heuristic algorithm is Simulated Annealing (SA) [25]. SA derives from the Monte Carlo method for statistically searching global minima. The method arises from a thermodynamic analogy. Specifically, it simulates the way that liquids freeze and crystallize, or metals cool and anneal. That is, at high temperatures atoms or molecules move freely with respect to one another. If the system is cooled slowly, thermal mobility is lost and the atoms or molecules can line themselves up and form a pure crystal that is completely ordered. This crystal is the state of minimum energy for this system. Here, we focus in SA for scheduling purposes. This approach has been tested in different environments like computational Grids [13][32]. SA is organized in several steps. First, a simulated annealing algorithm needs an initial solution, which is constructed by assigning at random a resource to each task. The annealing process runs through a number of iterations at each "temperature" to sample the search space. At each iteration, it generates a new solution by applying a random change on the current solution. Whether or not the new solution is accepted as a current solution is determined by the Metropolis criteria [25]. Once a specified number of iterations have

been completed, the system is in "equilibrium", i.e., a population has been generated obeying a Boltzmann statistical distribution. Then, the temperature is decreased at a specified rate. The process is repeated until the lowest allowed temperature has been reached. During this process, the algorithm keeps the best solution found, returning the last value as the final optimal solution.

In Tabu Search (TS) [12][15] some historical information related to the evolution of the search is kept. This is basically the itinerary through the visited solutions. Such information is used to guide the movement from one solution to the next, avoiding cycling. This is one of the most important features of the algorithm. The algorithm starts from an initial solution, typically a random one. At any iteration it has to find a new solution by making local movements over the current solution. The next solution is the best among all (or a subset of) possible solutions in the neighborhood. To carry out the exploration process, recently visited solutions should be avoided (tabu list). Therefore, once a solution is visited the movement from which it was obtained is considered tabu. Note that the neighborhood of the solutions will be changing along the exploration. So, in a certain sense we have a dynamic neighborhood. Typically, there are two kinds of tabu lists. On one hand, a long term memory maintains the history through all the exploration process as a whole. On the other, a short term memory keeps the most recently visited tabu movements. A movement with a tabu status (tabu movement) is avoided unless it satisfies certain aspiration conditions. This prevents falling into local minima. There are two kinds of stopping conditions in Tabu Search. The first relates to the Tabu Search as a whole (when the algorithm finishes). The second is a stopping condition over the search of the best among all solutions in the neighborhood.

As presented above, the QSS and ESS self-scheduling algorithms depend on three and two parameters respectively. These parameters determine the behavior of the algorithms. Therefore, for a given computational environment it is necessary to select the most appropriate (optimal) values of these parameters to obtain a good load balance and to minimize the overall computation time. In previous studies, we obtained the best parameters from experimental measures on an actual system [8][9]. However, this is a slow and hard process that we would repeat each time the execution environment changes. In these conditions, the systematic exploration of the parameter space when several (more than two) parameters do exist is simply unmanageable.

Previously, we presented in [1] a way to obtain optimal QSS and ESS parameters using a heuristic approach. To such an end, we simulated the execution environment (a computational Grid in our case). So, using the simulation, we could obtain the computational time of each algorithm for a given value of its parameters. Therefore, it was possible to apply a heuristic algorithm to explore the behavior of the scheduling method for different values of the parameters, minimizing the overall computation time. The heuristic algorithm selected was Simulated Annealing (SA).

Until now, we have tested these algorithms using a random workload distribution. Although, a priori, this is the most difficult case to schedule, it is interesting to observe the behaviour of our algorithms when they have to schedule other workload distributions. We have to consider that self-scheduling algorithms distribute the tasks into chunks in a decreasing way and therefore the load balance can be different depending on the task duration distributions. Considering as starting point the heuristic approach mentioned before, we study here its scheduling performance for the different workload distributions, see Figure 1.

In the next Section, we present an overview of the QSS and ESS self-scheduling algorithms, as well as the methodology used for their optimization in the different workload distributions considered. Section 3 presents and interprets the results found in the optimization process for each workload distribution. Finally, in Section 4 we present the main conclusions of this paper, and the perspectives for future works.

2. Methodology

In this work the Quadratic Self-Scheduling (QSS) and Exponential Self-Scheduling (QSS) algorithms are used as basic scheduling strategies. QSS [8][9] is based on a Taylor expansion of the chunks distribution function, C(t), limited to the quadratic term. Therefore, QSS is given by

$$C(t) = a + bt + ct^2 \tag{1}$$

where t represents the t-th chunk assigned to a processor. To apply QSS we need the a, b and c coefficients of equation (1). Thus, assuming that the quadratic form is a good approach to C(t), we can select three reference points (C(t), t) and solve for the resulting system of equations. Useful points are $(C_0, 0)$, ($C_{N/2}$, N/2) and (C_N, N), where N is the total number of chunks. Solving for a, b and c, we obtain,

$$\begin{aligned} a &= C_0 \\ b &= (4C_{N/2} - C_N - 3C_0)/N \\ c &= (2C_0 + 2C_N - 4C_{N/2})/N^2 \end{aligned} \tag{2}$$

where N is defined [8] by,

$$N = 6I/(4C_{N/2} + C_N + C_0) \tag{3}$$

being I the total number of tasks.

The $C_{N/2}$ value is given by,

$$C_{N/2} = \frac{C_N + C_0}{\delta} \tag{4}$$

where δ is a parameter. Assuming C_0 and C_N fixed, the $C_{N/2}$ value determines the slope of equation (1) at a given point. Therefore, depending on δ, the slope of the quadratic function for a t value is higher or smaller than that of the linear case, which corresponds to δ=2. These two cases are defined by

$$\begin{aligned} Case\ a \Rightarrow C_{N/2} > \frac{C_0 + C_N}{2} \ \Rightarrow \frac{d^2C(t)}{dt^2} < 0 \\ Case\ b \Rightarrow C_{N/2} < \frac{C_0 + C_N}{2} \ \Rightarrow \frac{d^2C(t)}{dt^2} > 0 \end{aligned} \tag{5}$$

The two cases are compared with the linear model in Figure 2. As we can see, case a) is a function concave down. Here $N_q < N_l$, where the q and l subscripts are used to distinguish the quadratic from the linear model, respectively. This smaller number of chunks in the QSS algorithm translates in a smaller communication overhead to the expense of higher initial chunk sizes, see Figure 2. On the other hand, it is possible to invert this behaviour, as shown by the dashed line in Figure 2 (case b), where the function is concave up. Here, chunk sizes are smaller than in the linear case (almost all the way) to the expense of a higher number of chunks, $N_q' > N_l$. In this case, we have a better load balance, but the communication overhead increases. Therefore, the QSS algorithm can be tuned to optimize the ratio of load balance to overhead by selecting an appropriate value of the $C_{N/2}$ coefficient.

In the present work, the values of the three parameters, C_0, C_N and δ are heuristically optimized in the simulated execution environment.

On the other hand, we have Exponential Self-Scheduling (ESS) [9]. ESS belongs to the family of algorithms based in the slope of the chunks distribution function, C(t). So, we consider that the rate of variation of C(t) is a decreasing function of t, g(t). Therefore, we have the general expression,

$$\frac{dC(t)}{dt} = g(t) \tag{6}$$

Equation (6) defines a differential equation. After integration we will have an explicit functional form for C(t) as a function of t.

ESS considers that the slope (negative) is proportional to the chunk size,

$$\frac{dC(t)}{dt} = -kC(t) \tag{7}$$

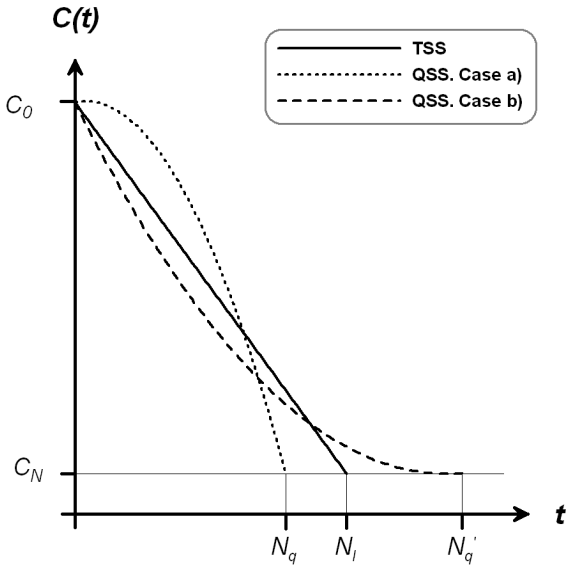


Figure 2. Evolution of the $C(t)$ function for the linear case (TSS) and QSS algorithms as a function of the number of chunks, t.

Here, k is a parameter and t represents the t-th chunk assigned to a processor. Equation (7) can be integrated by separation of variables yielding [9],

$$C(t) = C_0 e^{-kt} \tag{8}$$

Equation (8) defines the Exponential Self-Scheduling (ESS) algorithm. Here, C_0 and k are the parameters to be optimized in the simulated working environment.

With respect to the SA heuristic, as applied here, we consider that the function to minimize, the cost function (f), is the overall computation time needed to process a set of tasks, i.e., its makespan. In turn, we consider that the cost function depends on s, the set of parameters used by each self-scheduling algorithm. The corresponding pseudocode is shown in Chart 1.

Here, T is the system's "temperature" defined as a given value of the cost function, f(s). This is initialized to a high value. Symbols s and s' represent sets of the scheduling algorithm parameters. Three parameters are needed for QSS, and two for ESS. The f(s) function represents the cost of the s solution, which is initialized to a high value. As cooling schedule, we use an exponential approach $T_{n+1} = rT_n$, with $r = 0.989$ [28]. The total number of iterations performed for each temperature (equilibration) is given by L. In our case, after a previous calibration, we have fixed L to 200 iterations. Finally, FROZEN is the lowest allowed temperature for the system. We select a very small value for FROZEN, $12\text{x}10^{-11}$, which has shown to give consistent results in SA calibration tests.

Chart 1. Simulated Annealing Pseudocode

```
Function SimulatedAnnealing()
begin
  T := initial "temperature"
  s := initial parameters, S0
  repeat
    for i := 1 to L do begin
      generates new parameters s'
      if f(s') < f(s) then
        s := s'
      else begin
        x := f(s')-f(s);
        if e^{-x/T} > random(0,1) then
          s:= s'
      end
    end
    T := r T
  until FROZEN
  return s
end
```

The cost function f(s) is obtained as the simulated time (in seconds) necessary to solve all tasks in the specified execution environment. The tasks are scheduled according to QSS or ESS, and the optimal parameters are given by the final value of s in Chart 1. The simulator is organized as follows. Each task has associated a value, from 1 to 10, which represents its duration (in seconds). Task durations are randomly generated except when we want to test a uniform workload. In this case, the duration of each task is 5. On the other hand, in the increasing and decreasing cases the task durations are shorted in the required order. As previously commented, QSS and ESS allocate sets of tasks (chunks). So, the duration of a chunk is the sum of all tasks durations composing it. The computing (CPU) time for a chunk is calculated dividing its duration by the relative computing power of the processor where the chunk is executed. This computing power is referred to the fastest processor. Thus, lower values correspond to slower processors. To this value we add the temporal cost of transferring the chunk to the processor where it is executed. In addition, the scheduling cost introduced by the local queuing software is included as well.

The execution environment represented in the simulation is an Internet-Based Grid of computers [14]. Therefore, it is composed by two main components: network links and computer elements. Network links have associated a value that represents the temporal cost, in seconds, of transferring a file between two machines through Internet. This value is obtained as an approximate estimate of the transport latency [18]. To such an end, we consider actual measures of the bandwidth (bw) and of the smoothed round trip time (srtt) [21]. This last estimates future round trip times by sampling the behavior of packets sent over a connection and averaging those samples. Half the srtt is used as a rough estimate of an effective time of flight. So, the temporal cost of a network link (Tt) is given by $Tt = (file\_size/bw) + srtt/2$, where "file_size" represents the physical size of the chunk being sent. We consider that all chunks have the same physical size. On the other hand, a computer element is typically a computer cluster [5]. This is composed by a number of processors connected through an internal network. So, each simulated computer element has an associated array, which collects the relative computing power (a real number) for its processors (CPUs). The computer power of each processor is determined experimentally. The temporal cost of the internal network is considered negligible with respect to the temporal cost of the Internet network links.

The simulated execution environment used is a replica of the computational Grid considered in [9]. In Figure 3, we can see the execution environment. The Grid is made up by a client (qcycar) and three computer elements (C.E.): Hermes, Tales and Popocatepetl (Popo). Hermes and Tales are placed in Ciudad Real (Spain), and they are composed by 8 processors each. On the other hand, Popo is placed in Puebla (Mexico), and it is composed by 4 processors. We have subdivided the environment characteristics into three parts: network, processors and scheduling cost. The network characteristics, for each computer element, are collected in Table 1. The processors characteristics are shown in Table 2. The scheduling cost could be attributed to the software. After several tests, we have observed that the queue managers introduce the most important software delay. In our case, this delay is determined to be about 0.5 seconds.

Table 1. Network Characteristics. bw represents the bandwidth and srtt the smoothed round trip time

C.E.	Network	
Hermes	bw = 94.8 Mb/s	srtt = 241 μs
Tales	bw = 94.8 Mb/s	srtt = 241 μs
Popo	bw = 243 Kb/s	srtt = 665.535 ms

In this work, we perform several tests to verify the effect of SA optimized parameters in the efficiency of QSS and ESS. Tests with 1000, 2000, 5000 and 10000 tasks are considered. To determine the influence of the physical chunk size, these tests have been performed twice using a random workload. In the first case, we consider a physical chunk size of 1 Mb. In the second, we increase the chunk size to 10 Mb. The optimal values for the QSS and ESS parameters obtained by SA are used to compare the behavior of QSS and ESS. Moreover, we compare these results against

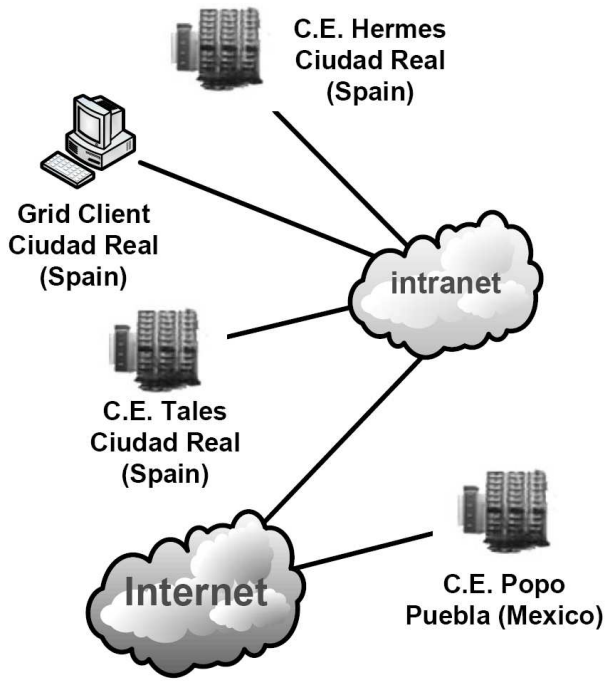


Figure 3. Execution Environment Simulated for the different tests: Internet based Grid of Computers.

the results obtained using the parameters experimentally determined for QSS and ESS in [9]. Finally, we compare the behaviour of the algorithms in the four workload distributions collected in Figure 1. In each case, new optimal QSS and ESS parameters are obtained. On the other hand, the increasing and decreasing workloads are obtained generating random tasks, and sorting them in the correct way. Each test is performed 100 times to obtain average results and determine standard deviations.

3. Results

First, we have performed different tests, using a random workload, to obtain the SA optimal parameters for QSS and ESS. Table 3 collects the results of the different tests, including the standard deviation (σ). In all cases, we observe a small σ. This implies that all the simulated values of the cost function are close to the average. Therefore, they can be considered very reliable. The main source for σ is the random generation of task durations in each simulation. Table 3 shows that the cost function increases linearly with the number of tasks. Comparing QSS and ESS, we see that both exhibit a similar performance. This result agrees with the data obtained in the experimental tests performed in [9].

To analyse the effect of the chunk sizes in the performance, we have repeated the simulations using a chunk size of 10Mb. Table 4 collects the results. We observe that the new values of the cost function, although higher as they must be, are very similar to those of Table 3. These results show that the present scheduling heuristic approach can compensate efficiently for changes of the network performance in a Grid system.

Table 2. Processors Characteristics For each computer element (C.E.) The number of processors of each type is specified. The relative computing power (R.P.) is also included

C.E.	Processors		
Hermes	**5** x P4 3.0 GHz		**3** x P4 2.4 GHz
R.P.	1		0.515
Tales	**1** x P4 3.0 GHz	**4** x P4 2.8 GHz	**3** x P4 2.4 GHz
R.P.	1	0.585	0.515
Popo	**4** x AMD64 1.6 GHz		
R.P.	0.448		

Table 3. Average cost function (cost) and standard deviation (σ) for QSS and ESS as a function of different number of tasks, in columns. The cost represents the simulated time, in seconds, necessary to solve all tasks. A physical chunk size of 1Mb is used.

QSS	1000	2000	5000	10000
Cost	428.830	846.191	2098.992	4179.476
σ	6.576	8.194	12.672	19.489
ESS	1000	2000	5000	10000
Cost	428.924	846.571	2098.921	4183.082
σ	6.213	8.817	15.341	17.586

Using SA, we obtained the minimum cost, total computing time, of QSS and ESS for the different test cases when the workload distribution is random. It would be interesting to compare these results against those obtained by using the QSS and ESS parameters experimentally determined in an actual Grid, see [9]. For QSS, these parameters were $C_0 = I/2P$, $\delta = 3$ and $C_N = 2$, where I is the total number of tasks and P is the number of processors. With these values, we have simulated the behavior of QSS obtaining the cost function, total computing time, of each test

Table 4. Average cost function (cost) and standard deviation (σ) for QSS and ESS as a function of different number of tasks, in columns. The cost represents the simulated time, in seconds, necessary to solve all tasks. A physical chunk size of 10Mb is used.

QSS	1000	2000	5000	10000
Cost	441.256	862.855	2120.979	4211.108
σ	7.183	8.399	15.672	19.253
ESS	1000	2000	5000	10000
Cost	445.563	870.002	2129.203	4211.697
σ	6.143	9.569	16.597	22.767

case. Since we have shown previously that the chunk size is not very significant, we only use a physical chunk size of 1Mb. Figure 4 collects the QSS results. We observe that the cost associated to the SA optimized parameters, "QSS SA", is always lower than that associated to the experimentally optimized QSS, "QSS E". In particular, we find that SA allows for an improvement between 3% and 6%. This is an interesting result, since SA obtains the most appropriate parameter values in 12 to 60 seconds, whereas the experimental values need a lengthy time consuming procedure on the actual Grid system [9].

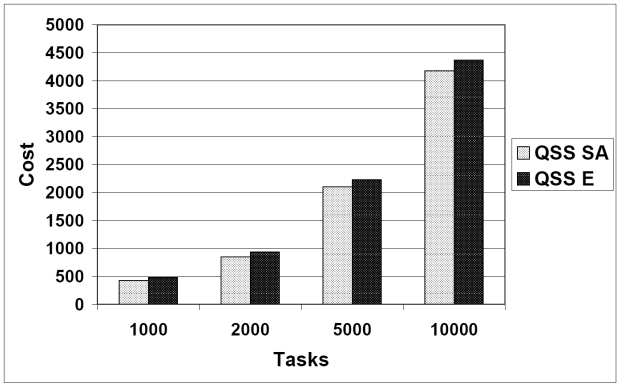


Figure 4. Comparison between the cost of QSS optimized using SA (QSS SA) and the cost of QSS optimized experimentally (QSS E). The cost is given in seconds.

With respect to ESS, the experimental parameters found in [9] are $Co = I/2P$ and $k = 0.017$. As in the QSS case, we have simulated the ESS behavior in our test cases using these values. Figure 5 shows graphically a comparison of the ESS results. We can appreciate that the cost of ESS with SA, "ESS SA", is always lower than that for the experimental parameters, "ESS E". In this case, simulated annealing gives us an improvement between 9% and 12%. Now, only 15 to 45 seconds are needed by SA to obtain the optimal results.

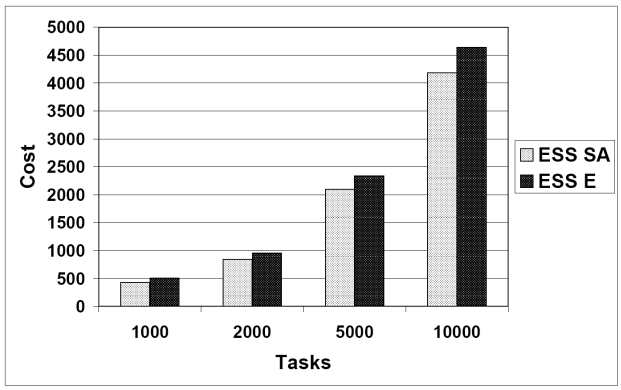


Figure 5. Comparison between the cost of ESS optimized using SA (ESS SA) and the cost of ESS optimized experimentally (ESS E). The cost is given in seconds.

The previous experiments were done using a random workload distribution. Usually, this kind of workload is the most difficult to handle, since the behaviour is unpredictable and therefore, it is more difficult to guarantee a good load balance. However, it is important to assess the behaviour of our algorithms when they have to schedule the remaining workloads: uniform, increasing, and decreasing (see Figure1). We have seen that the scheduling results of these workloads follows the results of the random one, with the QSS and ESS algorithms outperforming the rest of them. Therefore, here it is only exposed the results of the tests performed for QSS and ESS.

First, we have performed the tests using the parameters obtained experimentally. Thus, Figure 6 and Figure 7 show graphically the behaviour of QSS and ESS algorithms with the different workloads and the different test cases (number of tasks represented in x axis). The y axis represents the computational cost in seconds. We can observe that both QSS and ESS have a similar behaviour when scheduling different kinds of workloads. In both algorithms, the worst case corresponds to the increasing workload distribution. This is because self-scheduling algorithms distribute the tasks into chunks in a decreasing way. Therefore, too large tasks at the end tend to cause load imbalance. On the other hand, the decreasing workload have a good performance, even better than the random case. Here, there is too much workload at the start, but the algorithms can obtain a good load balance because they have enough time to

guarantee it. Nevertheless, the maximum difference in the execution time of the workloads is around a 12%. Finally, the uniform case is the less time consuming (between a 9% and a 18 % less). This is because all tasks have the same computational cost and because an uniforme workload is easier to schedule.

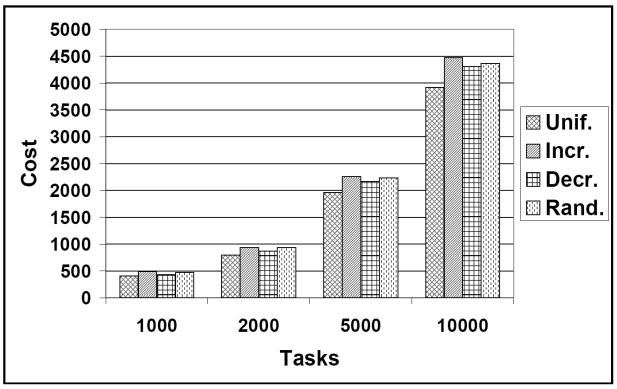


Figure 6. Behaviour of "QSS E" for different workload distributions. The QSS parameters were optimized experimentally (QSS E). The cost is given in seconds.

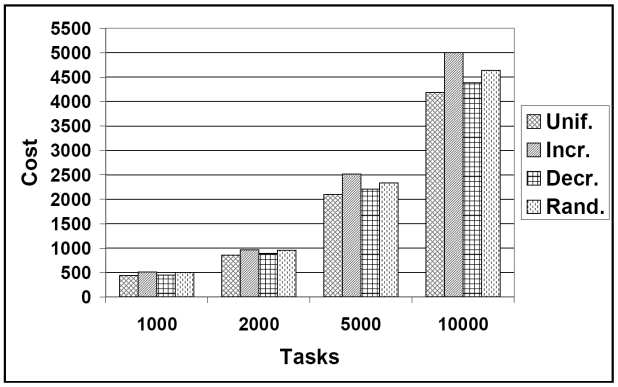


Figure 7. Behaviour of "ESS E" for different workload distributions. The ESS parameters were optimized experimentally (ESS E). The cost is given in seconds.

Now, for each case, we consider the behaviour of QSS and ESS after optimizing their parameters with the heuristic approach. Figure 8 and Figure 9 show the behaviour of both algorithms. We can observe that the uniform workload is again the less time consuming (around a 9%). On the other hand, the other workloads are executed more or less in the same time. So much so that the maximum difference in the execution time, among these workloads, is 1.3%. Therefore, even for different workloads and different number of tasks the heuristic approach provides equivalent performance.

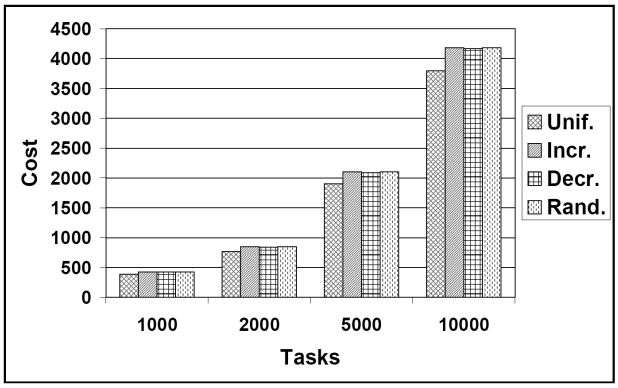


Figure 8. Behaviour of "QSS SA" for different workload distributions. The QSS parameters were optimized with SA (QSS SA). The cost is given in seconds.

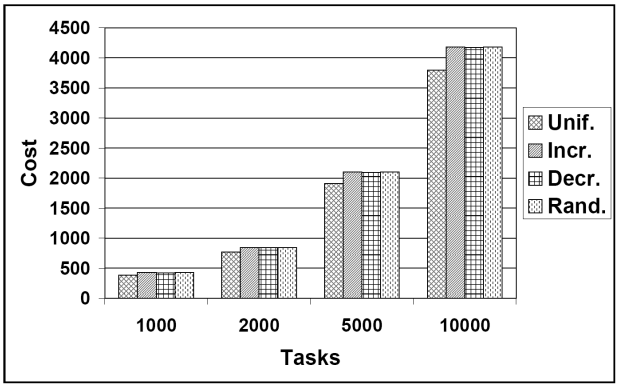


Figure 9. Behaviour of "ESS SA" for different workload distributions. The ESS parameters were optimized with SA (ESS SA). The cost is given in seconds.

4. Conclusion and Future Work

We have tackled the problem of scheduling a set of independent tasks in a computational Grid using a simulator and a heuristic approach based in SA. Using the simulator we can model the behavior of scheduling algorithms in an actual environment, but in a much shorter time than in an experimental study. We consider several test cases formed by several thousand tasks. These test cases can be subdivided in four groups depending on their workload distri-

butions. Thus, random, uniform, increasing and decreasing workload distributions are considered. A comparison like this is important, because there are situations in which the duration of the tasks is not known. Therefore, the tasks can not be sorted to optimize the scheduling. Working with the previously proposed QSS and ESS self-scheduling algorithms, we observe that optimizing their parameters using SA permits to reduce the overall computing time up to a 16%. This maximum difference is found in the increasing workload case. Moreover, after this optimization, we observe that the increasing, decreasing and random workloads have, more or less, the same overall execution time. These results show the high adaptability of our self-scheduling algorithms, which can be fully exploited thanks to the heuristic approach here presented.

The heuristic approach allows a reduction of the overall computing time. In addition, the time needed to obtain the optimal SA parameters for QSS and ESS, in the simulated environment, is negligible compared with the time needed for an experimental calibration in an actual Grid system. Furthermore, SA can optimize all the parameters in the scheduling algorithms, despite its number. In the general case, this is not possible using experimental measures. The test cases also show that the present heuristic approach is very efficient. In fact, we observe a simple linear increase of the execution time with the problem size.

Several enhancements can be devised for this preliminary study of a SA Grid scheduler. First, in the present study the characteristics of the Grid environment are considered static. A logical extension would be the scheduler to check, as a function of time, the state of the different network links and the performance of the available processors in the execution environment. Then, the simulated annealing procedure would obtain the most appropriate parameters for this specific behavior of the environment. Second, in the present work the chunk size is considered constant. This is an acceptable approach for testing and comparing the efficiency of scheduling algorithms in similar conditions. However, the situation is different in actual computations, where there are different physical chunk sizes. These sizes depend on the number of tasks making up the chunk. Considering this effect will permit to describe the different transfer costs of different chunks.

Acknowledgment

This work has been cofinanced by FEDER funds, the Consejería de Educación y Ciencia de la Junta de Comunidades de Castilla-La Mancha (grant # PBI08-0008), and the fellowship associated to the grant # PBI-05-009. The Ministerio de Educación y Ciencia (grant # FIS2005-00293) and the Universidad de Castilla-La Mancha are also acknowledged. The authors wish also to thank the Facultad de Ciencias Químicas and the Laboratorio de Química Teórica of the Universidad Autónoma de Puebla (Mexico), for the use of the Popo cluster.

References

[1] J. Díaz, S. Reyes, C. Muñoz-Caro, and A. Niño, "A Heuristic Approach to Task Scheduling in Internet-based Grids of Computers," *2nd Int. Conf. on Advanced Engineering Computing and Applications in Sciences (ADVCOMP 2008)*, Valencia, Spain, 2008, pp. 110-116.

[2] M. Aggarwal, R. D. Kent and A. Ngom, "Genetic Algorithm Based Scheduler for Computational Grids," in *Proc. 19th Annu. Int. Symp. High Performance Computing Systems and Applications (HPCS'05)*, Guelph, Ontario Canada, 2005, pp.209-215.

[3] F. Berman, "High-performance schedulers," in *The Grid: Blueprint for a New Computing Infrastructure*, I. Foster and C. Kesselman, Eds. San Fransisco, CA: Morgan-Kaufmann, 1999, pp. 279-309.

[4] R. Braun, H. Siegel, N. Beck, L. Boloni, M. Maheswaran, A. Reuther, J. Robertson, M. Theys, B. Yao, D. Hensgen and R. Freund, "A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems," *J. Par. Dist. Com.*, vol. 61, no. 6, pp. 810-837, 2001.

[5] R. Buyya, *High Performance Cluster Computing: Architectures and Systems*. New Jersey: Prentice Hall, vol. 1, 1999.

[6] T. L. Casavant and J. G. Kuhl, "A Taxonomy of Scheduling in General-Purpose Distributed Computing," *IEEE Trans. Softw. Eng.*, vol. 14, no. 2, pp. 141-154, 1988.

[7] J. Díaz, S. Reyes, A. Niño and C. Muñoz-Caro, "Un Algoritmo Autoplanificador Cuadrático para Clusters Heterogéneos de Computadores," *XVII Jornadas de Paralelismo*, Albacete, Spain, 2006, pp. 379-382.

[8] J. Díaz, S. Reyes, A. Niño and C. Muñoz-Caro, "A Quadratic Self-Scheduling Algorithm for Heterogeneous Distributed Computing Systems," *Proc. 5th Int. Workshop Algorithms, Models and Tools for Parallel Computing Heterogeneous Networks* (HeteroPar '06), Barcelona, Spain, 2006, pp. 1-8.

[9] J. Díaz, S. Reyes, A. Niño, and C. Muñoz-Caro, "New Self-Scheduling Schemes for Internet-Based Grids of Computers," *1st Iberian Grid Infrastructure Conf.* (IBERGRID), Santiago de Compostela, Spain, 2007, pp. 184-195.

[10] J. Díaz, S. Reyes, A. Niño, C. Muñoz-Caro, "Nuevas Familias de Algoritmos de Self-Scheduling para la Planificación de Tareas en Grids de Computadores," *XVIII Jornadas de Paralelismo*, Zaragoza, Spain, 2007, pp. 423-430.

[11] F. Dong and S. G. Akl, "Scheduling algorithms for grid computing: State of the art and open problems," School of Computing, Queen's University, Kingston, Ontario, 2006.

[12] I. D. Falco, R. D. Balio, E. Tarantino and R. Vaccaro, "Improving search by incorporating evolution principles in parallel tabu search," *IEEE Conf. Evolutionary Computation*, vol. 2, 1994, pp. 823-828.

[13] S. Fidanova, "Simulated Annealing for Grid Scheduling Problem," *Proc. IEEE John Vincent Atanasoff 2006 Int. Symp. Modern Computing*, 2006, pp. 41-45.

[14] I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, San Francisco, CA: Morgan Kauffman Publishers, 1999.

[15] F. Glover and M. Laguna, *Tabu Search*, Boston, MA: Kluwer Academic Publishers, 1997.

[16] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Boston, MA: Addison-Wesley, 1989.

[17] B. Hamidzadeh, D. J. Lilja and Y. Atif, "Dynamic Scheduling Techniques for Heterogeneous Computing Systems," *Concurr.: Pract. Exp*, vol. 7, pp. 633-652, 1995.

[18] J. L. Hennessy and D. A. Patterson, *Computer Architecture. A Quantitative Approach*, 3th Ed., San Francisco, CA: Morgan Kaufmann Publishers, 2003.

[19] E. S. H. Hou, N. Ansari and H. Ren, "A Genetic Algorithm for Multiprocessor Scheduling," *IEEE Trans. Parallel Distrib. Syst.*, vol. 5, no. 2, pp. 113-120, Feb. 1994.

[20] J. Yu and R. Buyya, "Workflow Schdeduling Algorithms for Grid Computing," Grid Computing and Distrib. Systems. Lab., Univ. Melbourne, Australia, Tech. Rep., GRIDS-TR-2007-10, May 2007.

[21] P. Karn, and C. Partridge, "Improving round-trip time estimates in reliable transport protocols," *ACM Trans. Comput. Syst.*, vol. 9, no. 4, pp. 364-373, Nov. 1991.

[22] T. H. Kim, and J. M. Purtilo, "Load Balancing for Parallel Loops in Workstation Clusters," *Proc. Int. Conf. Parallel Processing*, vol. 3, 1996, pp. 182-189.

[23] S. Kim, and J. B. Weissman, "A Genetic Algorithm Based Approach for Scheduling Decomposable Data Grid Applications," *Proc. 2004 Int. Conf. Parallel Processing* (ICPP'04), Montreal, Quebec Canada, 2004, pp. 406-413.

[24] D. J. Lilja, "Exploiting the Parallelism Available in Loops," *IEEE Computer*, vol. 27, vo. 2, pp. 13-26, 1994.

[25] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equations of State Calculations by Fast Computing Machines," *J. Chem. Phys.*, vol. 21, pp. 1087-1091, 1953.

[26] S. Penmatsa, A. T. Chronopoulos, N. T. Karonis and B. Toonen, "Implementation of Distributed Loop Scheduling Schemes on the TeraGrid," *Proc. 21st IEEE Int. Parallel and Distrib. Proc. Symp. (IPDPS 2007), 4th High Performance Grid Computing Workshop*, 2007.

[27] T. Philip, and C. R. Das, "Evaluation of Loop Scheduling Algorithms on Distributed Memory Systems," *Proc. Int. Conf. Parallel and Distrib. Computing Systems*, Washinton DC, 1997.

[28] P. Salamon, P. Sibani, R. Frost, *Facts, Conjectures, and Improvements for Simulated Annealing*, Philadelphia: SIAM Monographs Mathematical Modeling and Computation, 2002.

[29] P. J. Sokolowski, D. Grosu and C. Xu, "Analysis of Performance Behaviors of Grid Connected Clusters," in *Performance Evaluation of Parallel and Distributed Systems*, M. Ould-khaoua, and G. Min, Eds. Hauppauge, NY: Nova Science Publishers, 2006.

[30] S. Song, Y. Kwok and K. Hwang, "Security-Driven Heuristics and A Fast Genetic Algorithm for Trusted Grid Job Scheduling," *Proc. 19th IEEE Int. Parallel and Distrib. Proc. Symp.* (IPDPS'05), Denver, Colorado USA, 2005, pp. 65-74.

[31] C.-T. Yang, and S.-C. Chang, "A Parallel Loop Self-Scheduling on Extremely Heterogeneous PC Clusters," *9th Workshop Compiler Techniques for High-Performance Computing*, Academia Sinica, Taiwan, 2003.

[32] A. YarKhan, and J. J. Dongarra, "Experiments with Scheduling Using Simulated Annealing in a Grid Environment," *Proc. 3rd Int. Workshop on Grid Computing*, Baltimore, MD, 2002, pp. 232-242.

A MATLAB GUI for the analysis and reconstruction of signal and image data of a SAFT-based 3D Ultrasound Computer Tomograph

Torsten Hopp, Gregor F. Schwarzenberg, Michael Zapf, Nicole V. Ruiter
Institute of Data Processing and Electronics, Forschungszentrum Karlsruhe
Hermann-von-Helmholtz-Platz 1, 76344 Eggenstein-Leopoldshafen, Germany
e-mail: {torsten.hopp, gregor.schwarzenberg, michael.zapf, nicole.ruiter}@ipe.fzk.de

***Abstract*—At Forschungszentrum Karlsruhe, a new imaging system for early diagnosis of breast cancer is currently developed. The 3D Ultrasound Computer Tomograph (USCT) consists of approximately 2000 ultrasound transducers, which produce 3.5 million A-scans (amplitude scans) summing up to 20 GB of raw data for one image. The large number of A-scans, the large amount of data and the complex relationship between raw data and reconstructed image makes analysis, understanding and further development difficult for the scientists and especially for new employees and students. For this reason, an interactive graphical user interface (GUI) was developed using MATLAB. It integrates existent analysis methods and is easily extendable with new functionality via a plugin concept. The software provides several visualization functions for the raw data, the reconstructed 3D images, the USCT aperture and the relationships between them. This approaches demonstrate that MATLAB is not only applicable as programming language for numerical problems, but also adequate for representing complex systems by a GUI. It has a large benefit for the working group as it is used as common development platform: The plugin concept is widely used to integrate new analysis methods and share them with the rest of the scientists. The GUI and the visualization of the complex relationships of the USCT reduces the training period for new employees and students. An evaluation of the usability shows that the users evaluated the user interface to be very helpful, clearly arranged and beneficial for a better understanding of the coherencies of the USCT system.**

MATLAB; Graphical User Interface; Ultrasound Computer Tomography; Analysis software

I. Introduction

At Forschungszentrum Karlsruhe, a 3D Ultrasound Computer Tomograph (USCT) is currently developed [1][2]. The long term goal of the system under construction is the development of a new imaging modality for early breast cancer detection. It produces images in significant higher quality than conventional sonography. In contrast to conventional (X-ray) computer tomography and mammography there is no radiation exposure for the patient.

The system consists of approximately 2000 transducers, arranged in layers on a cylindrical tank, which is filled with water. The tank can be rotated in small steps. For a measurement each emitter emits an unfocused ultrasound pulse, while all other receivers record the ultrasound reflections caused by the objects located in the cylinder. This procedure creates up to 3.5 million A-scans (amplitude scans), which is equal to more than 20 GB of raw data.

In the first step, the raw data is read out from the data acquisition hardware and stored in a MATLAB data format [3] on a file server. In the second step a reconstruction method based on SAFT (synthetic aperture focusing technique) [4] and implemented in MATLAB is used to calculate 3D images based on the raw data. This workflow including the three levels (USCT aperture, raw data and 3D images) is shown in Figure 1. The relationship between raw data and reconstructed 3D images is not intuitive due to the large number of A-scans and the 3D sensor geometry. Every A-scan as well as the reconstructed 3D images correspond to 3D positions in the USCT aperture. For the scientists it is necessary to get a comprehensive overview of this large amount of A-scans and the whole system.

So far most analysis tools are implemented as command line scripts using several parameters. These scripts are difficult to use especially for new employees and students joining the group.

There are several libraries and software tools implementing only parts of the required functionality. ImageJ is a Java based image editing and analysis software including a powerful plugin concept [5]. For 2D and 3D visualization several commercial software systems are available as e.g., Volume Graphics Studio [6]. For signal processing, MATLAB and LabVIEW [7] are commonly used. However, the different software systems are standalone applications and not aimed to work together. An approach using MATLAB for the implementation of a GUI for analysis of ocean acoustic propagation is described in [8], only focusing on this specific topic. To the knowledge of the authors, no software system satisfies the requirements of the USCT project for an integrated software for the analysis and exploration of signal and image data.

For this reason a software tool was developed which integrates the different levels of the USCT workflow into one interactive GUI [1]. Because of constant changes and additions during the development process of the new imaging method, the interactive GUI has to be adaptable and extendable to new functionality. As the scientists implement the USCT algorithms in MATLAB, the implementation of the interactive GUI was also carried out in MATLAB, despite its limited GUI

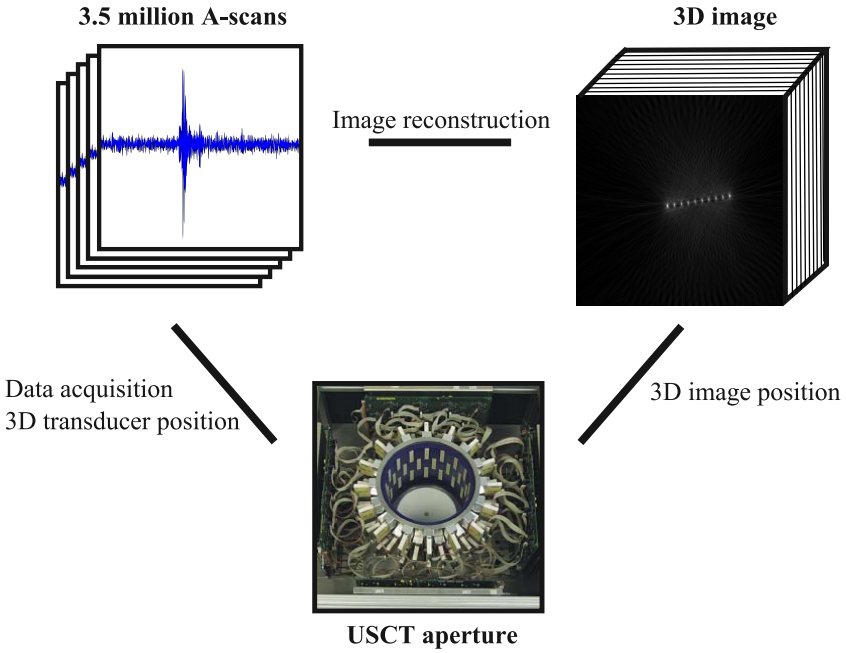


Figure 1. Workflow in the USCT: The USCT (bottom) produces A-scans (left), which are recorded by the data acquisition software. The reconstruction software creates 3D images (right) using the A-scans. The image corresponds to a 3D region in the USCT.

capabilities.

The aim of this paper is to give an outline of the design of the developed software and to show, how the limited GUI features of MATLAB can be used to create a complex interactive and adaptable GUI. In the following, an introduction to the ultrasound computer tomograph is given followed by an overview of the capabilities of the GUI elements that MATLAB can offer. After this, the design and structure of the developed software is described. Section IV details selected functionalities of the software. A user evaluation is shown in Section V and, finally, a conclusion is presented.

II. Basics

In this section, the basics on Ultrasound Computer Tomography are presented to motivate the need for a software system representing the workflow from raw data to reconstructed images. Furthermore, capabilities of MATLAB are shown, which form the framework used for the development of the software.

A. 3D Ultrasound Computer Tomography

Our first prototype for 3D Ultrasound Computer Tomography consists of a cylindrical tank with a diameter of 18 cm and a height of 15 cm. On the cylinder wall three layers of 16 Transducer Array Systems (TAS) [9] are mounted, each consisting of eight ultrasound emitters and 32 receivers resulting in 384 emitters and 1536 receivers (Figure 2). To achieve an uniform distribution of emitters and receivers, the cylinder can be rotated to six positions using a stepping motor. Applying six motor positions, the system consists of a virtual number of 2304 emitters and 9216 receivers, creating the 3.5 million A-scans. The emitters are virtually numbered in layers from 0 to 23 and elements from 0 to 95. The receivers are virtually numbered in layers from 0 to 47 and elements from 0 to 191.

A measurement is done by emitting an approximately spherically ultrasound pulse front (center frequency of 2.7 MHz) into the tank, which is filled with water as coupling medium. The ultrasound is absorbed, scattered and reflected, depending on the objects within the tank. Simultaneously, all receivers of the system record the amplitudes of the transmitted and reflected pulses. Afterwards the next emitter sends an ultrasound pulse while all other transducers receive. The data acquisition hardware of the system digitalizes at a sampling rate of 10 MHz, which results in A-scans of 3000 samples (each 300 μs). The complete procedure for all emitter-receiver-combinations or a subset of them is controlled by a data acquisition software (Andromeda) based on Java, which reads out the hardware memory via native libraries. The data is stored on a file server in a MATLAB data format. Additionally to the raw data, information about the measurement parameters (temperature, emitted pulse etc.) is stored in separate files.

The A-scans contain the transmitted pulse from emitter i at position $\vec{e}_i$ to receiver j at position $\vec{r}_j$ as well as the reflections from the object (Figure 3). With the information about the time the ultrasound needed from the emitter to the object and

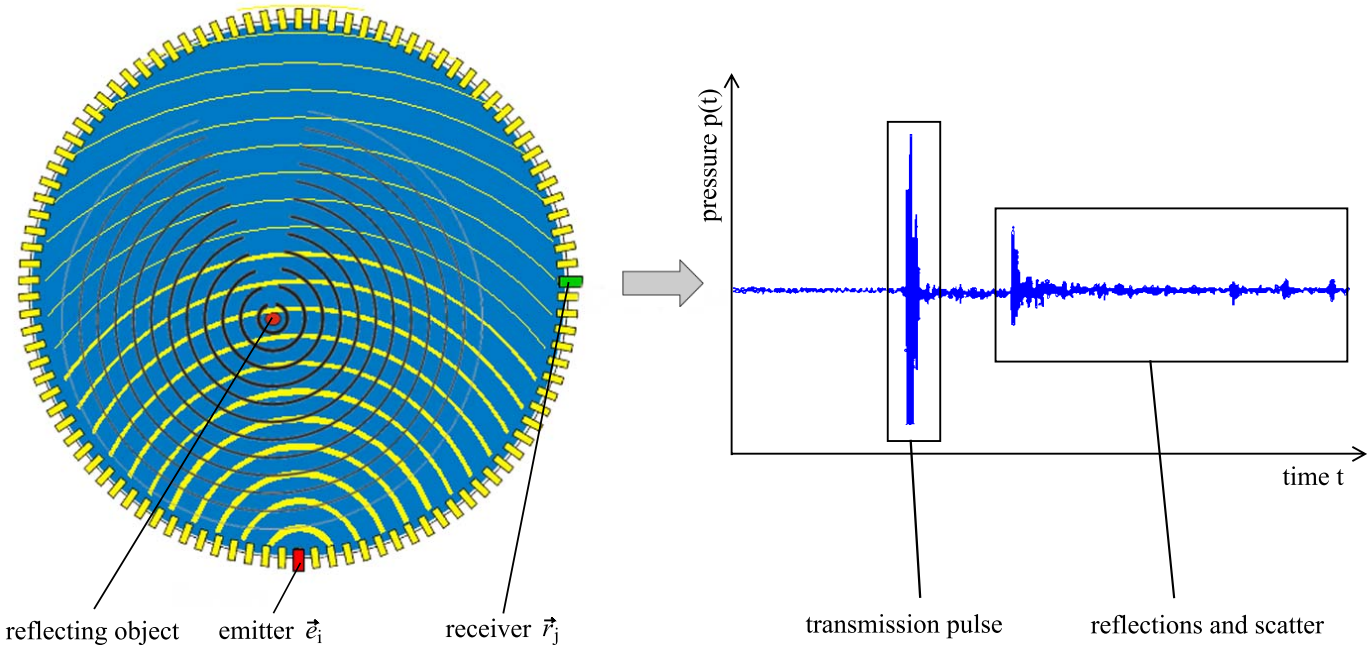


Figure 3. Dependence of A-scans from sensor geometry. Left: Principle of the data acquisition for USCT simplified in 2D. The emitter emits an ultrasound pulse, which is reflected by the object. Here an idealized point scatterer scatters the ultrasound in all directions. The reflected and transmitted signals are recorded by all the receivers. Right: A-scan recorded at green receiver as pressure over time. The first signal is the transmission pulse (directly transmitted ultrasound). The successive signals are reflections and scatter from the imaged object and the cylinder.

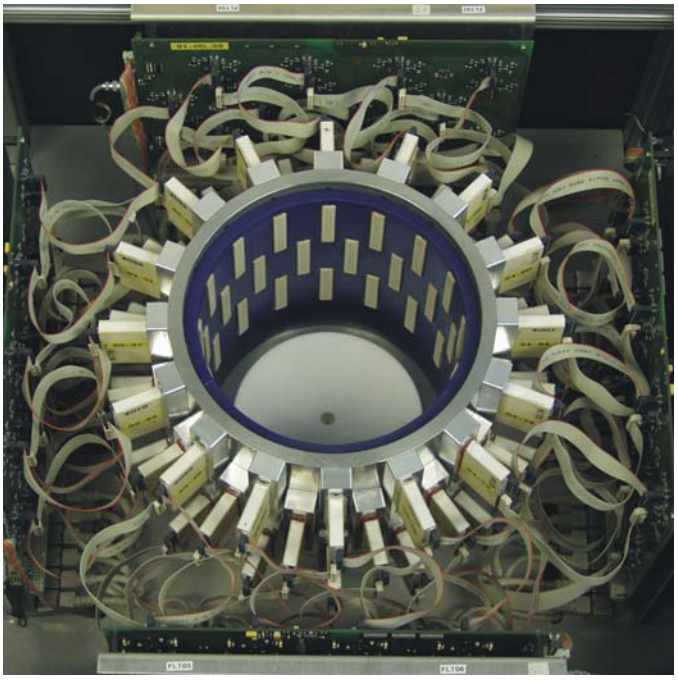


Figure 2. The prototype of a 3D Ultrasound Computer Tomograph: water tank (middle) equipped with 48 TAS which are connected by flat cables to four distribution boards (edges of the picture).

from the object to the receiver, a spheroidal shell with emitter and receiver as focal points representing all possible positions of the scattering object can be calculated. This assumes a constant speed of sound v throughout the water. The volume of interest $I(x)$ at the position $\vec{x}$ is then reconstructed by summing over each projection of each sample of an A-scan $A_{i,j}$ with its specific amplitude. Repeating this for all $m \cdot n$ emitter-receiver-combinations – hence for all A-scans – high values are summed up at points that are reflecting ultrasound at a high level (Figure 4, equation 1). This method is known amongst others as SAFT [4] and can be written as:

$$I(\vec{x}) = \sum_{i=1}^{m} \sum_{j=1}^{n} A_{i,j}(t),\ t = \left(\frac{\|\vec{e}_i - \vec{x}\| + \|\vec{r}_j - \vec{x}\|}{v} \right) \quad (1)$$

The image reconstruction software is subjected to constant development. The code is mainly written in MATLAB. The kernel of the backprojection was implemented in assembler for speed up [10]. Depending on the resolution the reconstruction of a complete volume can take from a few hours up to several weeks on a standard PC. The reconstruction of two dimensional slice images takes some seconds to a few minutes.

Currently a second prototype of the 3D Ultrasound Computer Tomograph is being developed. The sensor distribution was optimized to increase the illumination level as well as the contrast and theoretical resolution [11]. While the first prototype consists of a cylindrical water tank, the second generation changes the shape to a half ellipsoid. In the course of this new development new Transducer Array Systems are built and the second generation of the acquisition hardware promises first clinical experiments.

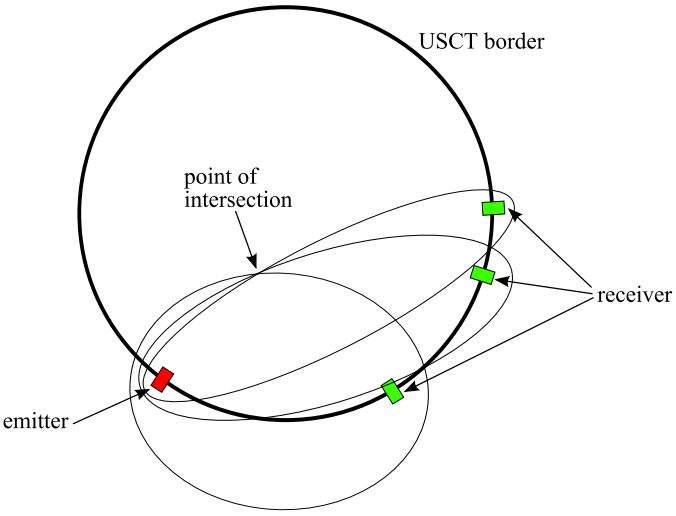


Figure 4. Principle of the SAFT. Ellipses for three emitter-receiver-combinations intersect in one point describing the position of the object.

B. MATLAB capabilities

MATLAB (the abbreviation stands for 'MATrix LABoratory') is a proprietary mathematical software developed by 'The Mathworks Inc.'. It provides a wide functionality for numerical calculations and allows the user to work with vectors and matrices in a straightforward way. The kernel of MATLAB is extendable with a large selection of toolboxes for different application domains. An overview of the available toolboxes can be found on the MATLAB website [12]. The main part of MATLAB interpreter, which interprets commands typed by the user. The commands derive from MATLAB's own programming language called M-Code. For rapid prototyping in signal and image processing, as necessary for improvement of the USCT reconstruction code, MATLAB provides a very handy platform due to the availability of many functions and toolboxes.

The MATLAB GUI functionality is not often used in scientific programming. Although the GUI functionality is not as powerful as e.g., the GUI functionality of Java (e.g., tabs are missing), it was the method of choice for this application, as the existing script software could easily be integrated. A complete reference for all interaction objects provided by MATLAB can be found in [13].

In MATLAB 6 only ten interaction objects were available to build GUIs:

- Textfields
- Edit Boxes
- Frames
- Pushbuttons
- Togglebuttons
- Checkboxes
- Radiobuttons
- Popup Menus
- List Boxes
- Sliders

Beside these interaction objects, MATLAB offers the powerful axes object used for data visualization. It can be extended by numerous functions for user interaction. Moreover every MATLAB GUI object can be appended by any desired data structure, which simplifies object-based interaction.

MATLAB offers several features for creating an interactive GUI, which kept the effort in development low. The extendability is based on the fact that MATLAB is an interpreter language. By just adding folders to the MATLAB path it is possible to access new functionality, even at runtime. For data visualization MATLAB uses a so-called *axes* object, which is able to visualize arbitrary 2D and 3D data. It offers the possibility to define click functions and context menus for all displayed graphical elements even in 3D. Moreover any form of data can be assigned individually to each graphics object. This simplifies object-based interaction and allows the easy design of interactive GUIs of any kind.

III. Software structure and design

In the following, the structure and design of the developed software is presented. Starting with the basic structure, the partition into three main parts is described. This is followed by presentation of the data mapping functions and the extendability of the software.

A. Basic structure

Due to the limited availability of MATLAB 7.x by the time of the development, MATLAB 6.1 was chosen as version for implementation. The downward compatibility and version checking of MATLAB ensures the usability of the interactive GUI with MATLAB 7.x.

To represent the imaging and image reconstruction in the USCT as concise as possible the functional requirements were divided into three parts:

1) *A-scan GUI*: contains all functions working directly on the A-scans.
2) *Images GUI*: contains all functionality for the reconstructed 3D images.
3) *USCT GUI*: contains the functionality for the representation of the USCT emitters and receivers for visualization purposes.

A main purpose of this representation in three parts is to show clearly what is done with the raw data when it is taken to reconstruct an image, to show how an image is built up by the raw data and where images and data sources are located in the USCT. The functionality covers recurrent procedures used to explore and analyze the experiments with the USCT.

The structure of the GUI for the system was designed to consider these requirements: the main GUI consists of three separate windows (MATLAB figures) related to the three parts of the functional requirements (Figure 5) This has the advantage that every part can also be used standalone. The challenge posed by the use of multi-figure GUIs is that they are complex to develop in MATLAB.

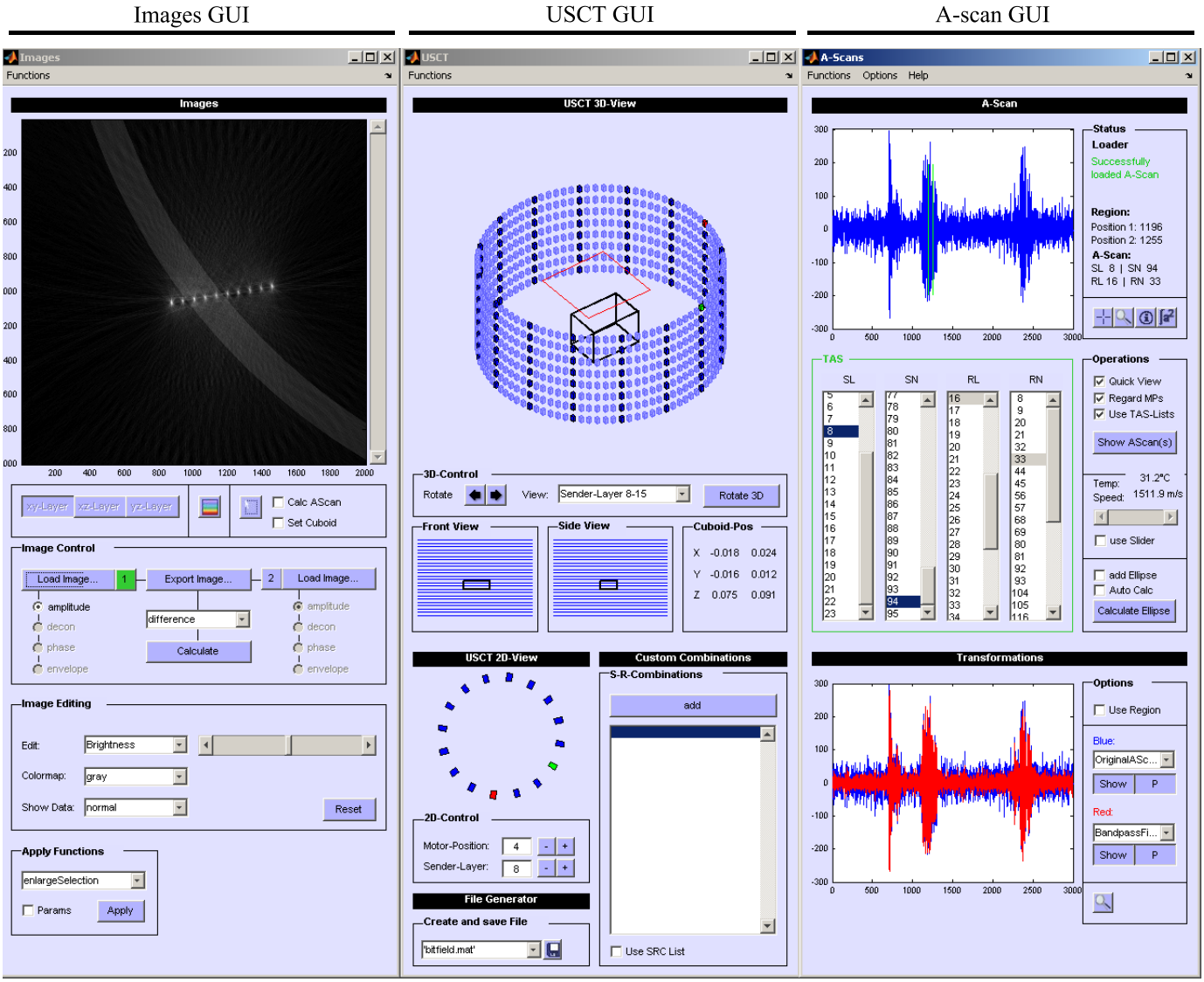


Figure 5. Graphical user interface with three separate windows: images related function at the left (*Images GUI*), USCT related functions in the middle (*USCT GUI*) and raw data related functions at the right (*A-scan GUI*).

B. A-scan GUI

The aim of the *A-scan GUI* is to give a quick overview of one or more A-scans from USCT experiments by selecting certain emitter-receiver-combinations. This is useful for quality control, signal-to-noise ratio and time-of-arrival analysis. This functionality is provided by four listboxes used to choose one or more emitter layers, emitter elements, receiver layers and receiver elements according to the USCT specification. Dependent on the number of selected A-scans the software plots the data to an *axes* object in the GUI or opens a scrollable dialog showing multiple plots for direct comparison (Figure 6). The scrollable dialog was done using a MATLAB script by Bjorn Gustavsson [14]. An additional functionality is to apply different signal processing functions to the A-scans without typing MATLAB commands. For this a second *axes* object is located at the lower half of the GUI. The user can choose from a long extendable list of transformations in a popup menu which are then applied to the currently loaded A-scan or a marked region of it. It is also possible to overlay a second graph for comparison of the original A-scan and transformed data. Measurement information that formerly had to be read out of separate files is now presented in a structured dialog reducing the effort of gathering information to a minimum of one click.

C. Images GUI

The *Images GUI* deals with large volume datasets. To give the user an overview, appropriate display functions are necessary. Therefore the *Images GUI* shows slices of 3D images in an *axes* object. The orientation of the slice can be chosen by the user as well as the slice number. A slider allows to browse through the volume. For comparison of images, which were reconstructed with different parameters, the user

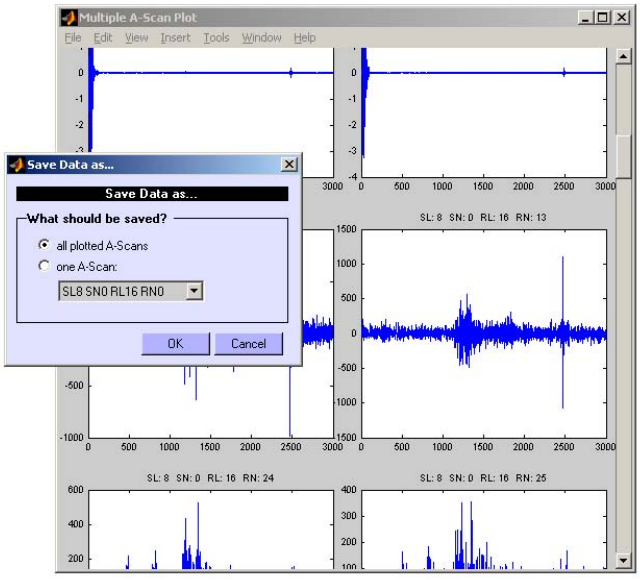


Figure 6. Scrollable dialog showing multiple plots of A-Scans for comparison.

can swap between two images and apply aggregation functions like calculating the difference of two images. This procedure is important for the analysis of different parameters for the reconstruction algorithm. To improve the appearance of the images, image processing functions are available. The images can be exported to the several graphical formats offered by MATLAB.

D. USCT GUI

A 3D model of the USCT aperture, i.e. emitter and receiver elements, is the main control element in the *USCT GUI*. It displays the connection between the *Images GUI* and the *A-scan GUI*, the interactive and color-coded visualization of selected transducer elements, and the location of currently loaded images. The model can be displayed in complete or in three parts according to the three TAS layers of the USCT. It can be rotated stepwise or freehand using the mouse.

Because of the usage of a parameter file the model can be adapted to all kinds of aperture geometries by simply exchanging this file. For the second generation of our 3D USCT prototype this will be of great advantage. For the the new developed geometry, a MATLAB file containing the cartesian transducer positions has to be built and copied into the specified folder of the USCT GUI.

Each element of the model represents an emitter and four receivers surrounding the emitter. By right-clicking an element a context menu offers functionality to mark the elements in different patterns as emitters or receivers. The left mouse button is programmed by the last chosen marking pattern and can be applied to other elements. Hence, the software is adaptive to click behavior of the user. The use of marking patterns and the programmable mouse button simplifies user interaction with the 384 elements shown in the model. An additional 2D model cutting out layers of the USCT also provides functionality for the marking of elements in a more precise way. The file generator uses the current marking pattern to create parameter files for the data acquisition and reconstruction software.

The markings in 3D are more intuitive for the scientists. They can be used to select a subset of A-scans in order to visualize them or calculate statistics.

E. Data mapping functions

For better understanding of the A-scan to image relationship, the system maps a region in a selected A-scan in the *A-scan GUI* to a currently loaded reconstructed image in the *Images GUI*. For the inverse process a region in the image can be calculated back to a time duration in the A-scan. Marked regions in an image are shown as a rectangle in the 3D model of the USCT giving the user an impression of the position of the currently shown slice image. The colored transducer elements in the *USCT GUI* and the illustration of the image position supports the understanding of the complex coherences. The data mapping functions allow the user a comfortable and easy to use in-depth analysis of the image formation process.

F. Extendability

The second fundamental purpose of the system was to provide extendability and adaptability. The interactive GUI serves as basis for the frequently used functions, which can be extended by new functionalities. Because of the ongoing development process modifications in all parts of the USCT workflow have to be introduced. New functions should be integrated into the system without major effort. Therefore a plugin concept was designed which allows the user to add new functionality by copying the source code – regarding the interface constraints – in defined folders. At startup the software checks the folders for new files and integrates them into pop-up menus or menus of the GUI for immediate access. The system offers interfaces for function integration. The interfaces can be grouped into:

- Standalone plugins without parameters and no return values. These plugins are used to integrate all kinds of tools into the software, as e.g., a standalone volume visualization tool . They are presented in the menu bar of each of the three windows. The GUI has to be designed by the user.
- Plugins using the currently loaded data of the software. They have input parameters and return defined data structures. Additionally to these defined parameters the programmer can add its own parameters for which a dynamically generated input dialog is created when calling the function. For these plugins the user does not need to program a GUI as it is generated automatically. This type of plugin is used for transformation functions on A-scans and for transformation and aggregation functions on images.

At program start the three parts of the GUI are arranged on the screen dependent on the adjusted screen resolution and all components including the computationally intensive 3D model are loaded in background while a status bar is displayed.

IV. Implementation of selected functions with respect to the MATLAB's GUI capabilities

Based on the basic overview on the software structure shown in the last section, the implementation of selected functions is presented in detail below in respect to MATLAB's GUI capabilities.

A. Main GUI

As described in the last section, the interactive GUI consists of three main windows which are realized as three MATLAB figures. The interaction between these independent figures is realized using MATLAB's global variables, which are available to all functions within a MATLAB session. At startup MATLAB stores all references to GUI objects in a specific variable. By declaring a global variable holding a copy of the references to GUI objects of the figure, the GUI objects can be accessed from every function or subfunction. To keep track of the numerous references every MATLAB figure (*A-scan GUI*, *Images GUI* and *USCT GUI*) creates its own global variable.

B. 3D USCT Model

A central interaction object of the system is the 3D model of the USCT. It is based on an USCT emitter geometry file which holds the position of every ultrasound emitter in cartesian coordinates. By plotting a voxel (created from six patches) at each position into the *axes* object the model is built up at startup of the software (Figure 7). This is done by use of the voxel function by Suresh Joel [15].

To restrict the number of voxels, one emitter and four receivers are represented as single voxel and each can be addressed by the corresponding emitter layer, emitter number and the according receiver layers and receiver numbers. These numbers are stored in each voxel using MATLAB's *UserData* variable, which is part of every GUI object. Within the assigned context menu of each voxel the user can select different options, e.g., setting the element as current emitter or receiver, setting the complete row or column as emitter or receiver and deselecting one, more or all elements. Depending on the users current choice the called routine reads its *UserData* variable via the voxel handle and sets the color of the voxel to red (emitter), green (receiver) or yellow (emitter and receiver). If the user sets an element as receiver an additional dialog is opened offering the selection of one to four of the receivers surrounding the selected element. The coloring routines are also called when selecting emitters or receivers in the *A-scan GUI*.

The name of the last callback routine is saved internally. By pressing the left mouse button which is pointing to a voxel this routine is called again. This results in an adaptive behavior of the left mouse button, which allows a fast marking of elements and avoids the repetitive use of the context menu. The 3D model can be rotated using the MATLAB rotating function for *axes* objects.

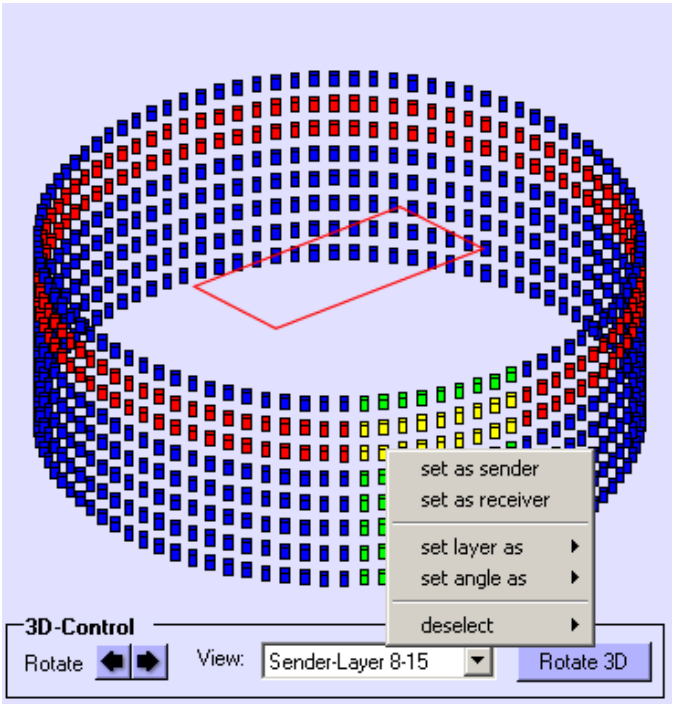


Figure 7. Interactive 3D model of the USCT with elements (blue) marked as emitter (red), receiver (green), emitter and receiver (yellow), the context menu called by clicking an element with the right mouse button and a red rectangle showing the position of a reconstructed image loaded in the *Images GUI*.

C. Data-image relationship

The relationship between raw data and reconstructed images is complex. For an in-depth analysis of reconstructed 3D images a method was needed to identify regions of one or more A-scans, which contribute to a specific part to the image. Also it is of interest which region of the image is concerned when selecting an area in an A-scan (e.g., a reflection). Therefore the data-image relationship can be visualized in both ways by functions marking a region in an A-scan by two points and in an image by a rectangle. This enables the user a detailed analysis of artifacts in the image or the spatial contribution of specific ultrasound pulses to the image.

Using MATLAB's *ginput* function two points restricting an A-scan to a relevant region can be chosen. The corresponding region in the currently displayed image slice is calculated by a routine based on the kernel of the reconstruction software. A slice image of a spheroidal hull, i.e. an ellipse, is calculated which is drawn into the reconstructed image. This can be done in two ways (Figure 8): The first method scales the pixel values of the calculated slice image to the colormap of the shown image and sums up both images. The second method creates an image by setting pixels within the ellipse to an opacity of 50 % and all other pixels to an opacity of 100 %. Moreover, these ellipses can be summed up, which demonstrates the process of the reconstruction for a number of A-scans.

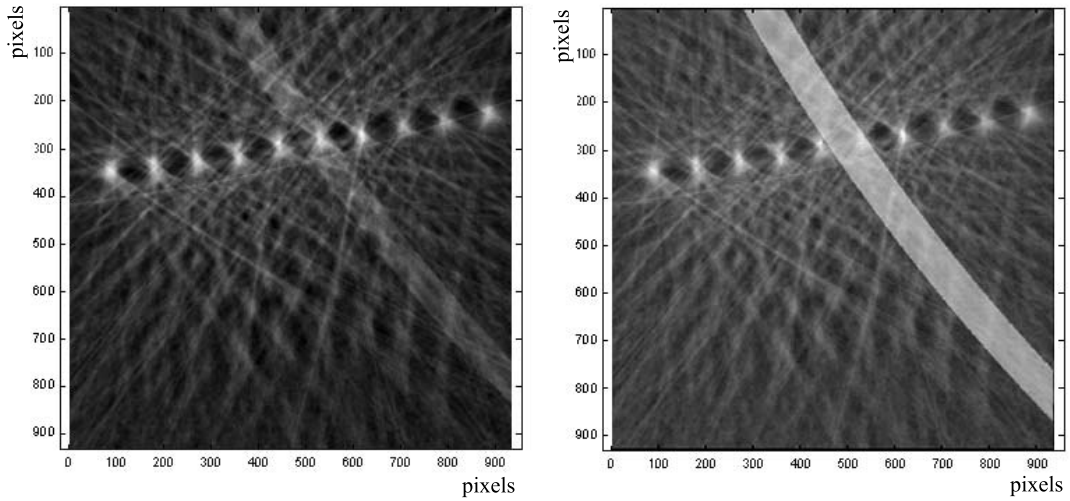


Figure 8. Slice of a reconstructed volume showing ten nylon threads with a diameter of 0.15 mm and spacing of 2 mm. Ellipses calculated by the system are drawn into the image - on the left by summing up the original and the ellipse image, on the right by using semi-transparent overlay.

Vice versa it can be of interest to know about the relationship of points or regions in the reconstructed images to A-scans. The system provides functionality to map an area R in the reconstructed image to a time duration in a chosen A-scan. For calculation of the two sample points t_{lower} and t_{upper} in the A-scan, which limit this region, the minimum and maximum distance from the chosen emitter to the chosen receiver via the object is calculated by the following terms regarding equation (1):

$$t_{lower} = \arg\min_{\vec{x}}(t), \vec{x} \in R \tag{2}$$

$$t_{upper} = \arg\max_{\vec{x}}(t), \vec{x} \in R \tag{3}$$

The resulting two sample points are drawn into the A-scan.

The calculation of the contributing part in an A-scan enables additional analysis functions such as the calculation of the N most contributing A-scans to a selected area in the reconstructed images. The routine searches for the highest amplitude in the computed region in each A-scan of a chosen subset and sorts them descending from the highest to the lowest peak. The resulting list is shown in a separate listfield, the emitters and receivers are marked in the USCT model. A-scans can be selected and chosen for further processing by A-scan analysis functions.

D. Extendability and plugins

The software has several interfaces for the extension with new functionality. When the software starts up the available functions are searched in predefined subfolders and entries are automatically generated in the according menus and pop-up menus of the GUI.

The plugins implementing the interface for standalone plugins can be called by accessing the menu entry in the particular figure. The software calls the specified function by the filename of the MATLAB source code file (.m-file). Information about the label of the plugin is stored in an identically named MATLAB data file (.mat-file) in the same folder.

The interfaces for non-standalone plugins have predefined input parameters and return a predefined data structure – e.g., an A-scan as input and return value, which is represented as an array of double values. If a function interface has more than these predefined parameters, an optional parameter can be passed as MATLAB struct which can contain arbitrary subvariables of different data types. The variable used as parameter is saved in a .mat-file identically named to the .m-file containing the function. When calling the function, the .mat-file reads its variable. Using the names of the subvariables within the struct and the default values defined for them, a dialog is created. The user can change these values which are then saved to the .mat-file (Figure 9). For better understanding of the parameters the .mat-file may contain an additional comment for each parameter which is shown to the user as tooltip. After saving modified parameters to the .mat-file the function is executed. The interface passes all parameters to it and passes back the result for visualization after execution.

This approach enables the encapsulation of any functionality by just two files where the .m-file contains the source code and the .mat-file holds the information for the GUI, the number of parameters, their default values and parameter descriptions. Due to the fact that MATLAB is an interpreter the .m-files can be easily accessed via adding them to the MATLAB path. It is possible to pass any data structure to the function, however, the dynamically created GUI can only process single numerical values so far.

The plugins implementing the interfaces can access all available data (e.g., the currently selected emitters and receivers)

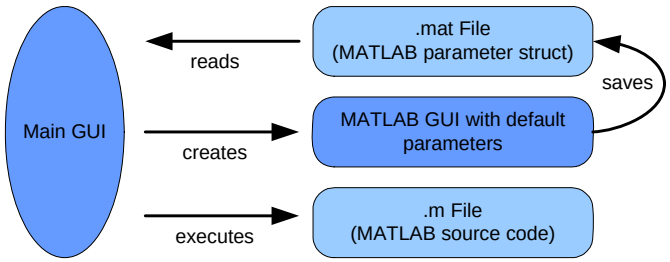


Figure 9. Files and dialogs used for the non-standalone plugins.

via a set of setter- and getter-functions, which encapsulate the access to the global variables and return the specific values.

E. Integration of the reconstruction software

Before the integration process, the reconstruction software was used as command line based tool and based on a simple text-file defining the various parameters for image reconstruction. In course of the integration, graphical user interfaces were developed giving the user a consistent view to all USCT software systems.

Therefore, the reconstruction software was integrated into the user interface by setting up a plugin calling the main function (*Image reconstruction GUI*). An additionally implemented GUI element in the *USCT GUI* offers selection of regions to be reconstructed (Figure 10). This was done using two axes objects for the representation of the emitter layers shown from two directions. Dragable lines refer to a cuboid drawn into the 3D USCT model shown in Section IV-B. The user can now select a region in 3D to retrieve the according cartesian coordinates of the edges, which will be used for definition of the region of interest to be reconstructed. They are automatically saved in the data structure of the reconstruction software, which defines all relevant parameters used for image reconstruction.

For changing parameters there is a dynamically generated GUI, the *Parameter GUI*, which reads out the parameter file of the reconstruction software. By specified types of input, it chooses interaction elements and arranges them in a MATLAB figure.

For a better overview, the inputs can be placed in groups according to their hierarchy level, which can be expanded and retracted by the use of push buttons. Since MATLAB does not offer a specific interaction element to manage such hierarchical structures, the rearrangement of the interaction elements, when expanding or retracting the parameter groups, has to be done via displacements of them. This is carried out by accessing the handles of following interaction elements in the parameter list and reset the position in the MATLAB figure window.

Another benefit of the *Parameter GUI* is an automatic validity check of the parameters. Also an effort was made to ensure downward compatibility for users working with the command line based tool. Therefore function wrappers were developed encapsulating the new functionality. Parsers had to be developed to read out the parameter text file and format it

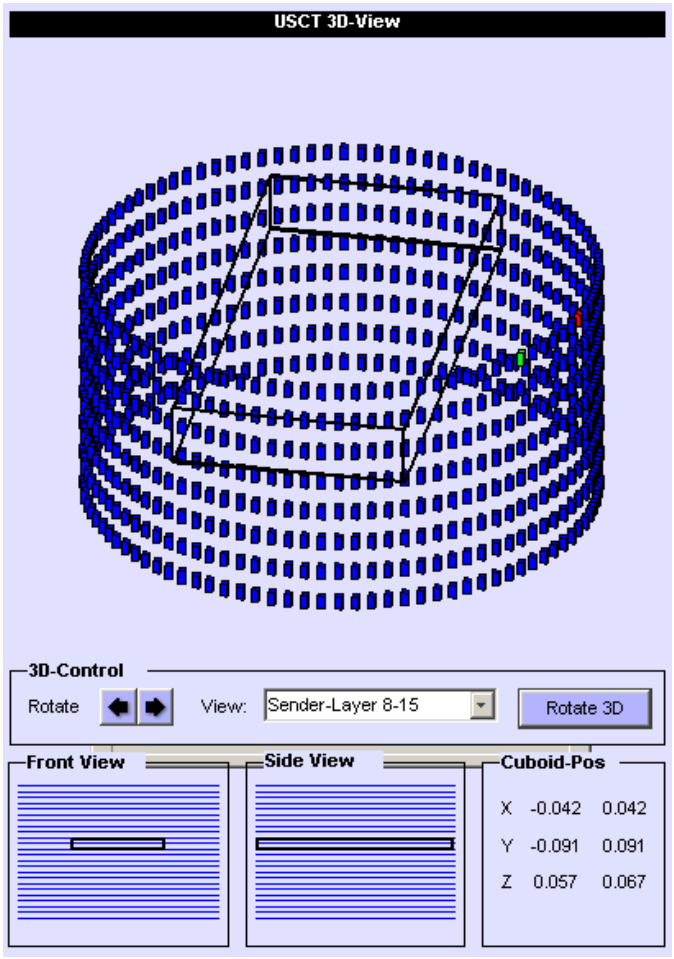


Figure 10. Selection of three dimensional regions to be reconstructed by the reconstruction software.

to an internal data structure used for the GUI generation. To stay consistent, values changed in the GUI have to be admitted to the text-based files by the use of text pattern matching.

Because of the dynamic generation of the GUI, it remains highly adaptable to changes in the reconstruction software.

Each parameter is stated by an importance value. When stating the category of the current user (beginners, intermediate and advanced) at program call, the importance value is applied to enable the appropriate parameters for the current user. E.g. beginners using the reconstruction software only get access to the most important parameters, while the rest of them are set by default values. This simplifies the complexity of the reconstruction for students and collaboration partners, which are just starting to use the reconstruction software and do their first image reconstructions. For experienced users all available parameters are shown. The visibility level can be set in the *Image reconstruction GUI* before calling the *Parameter GUI*.

V. Evaluation of Usability

An evaluation of the GUI was done using a usability questionnaire consisting of three parts. First, the user has to evaluate his know-how about ultrasound computer tomography

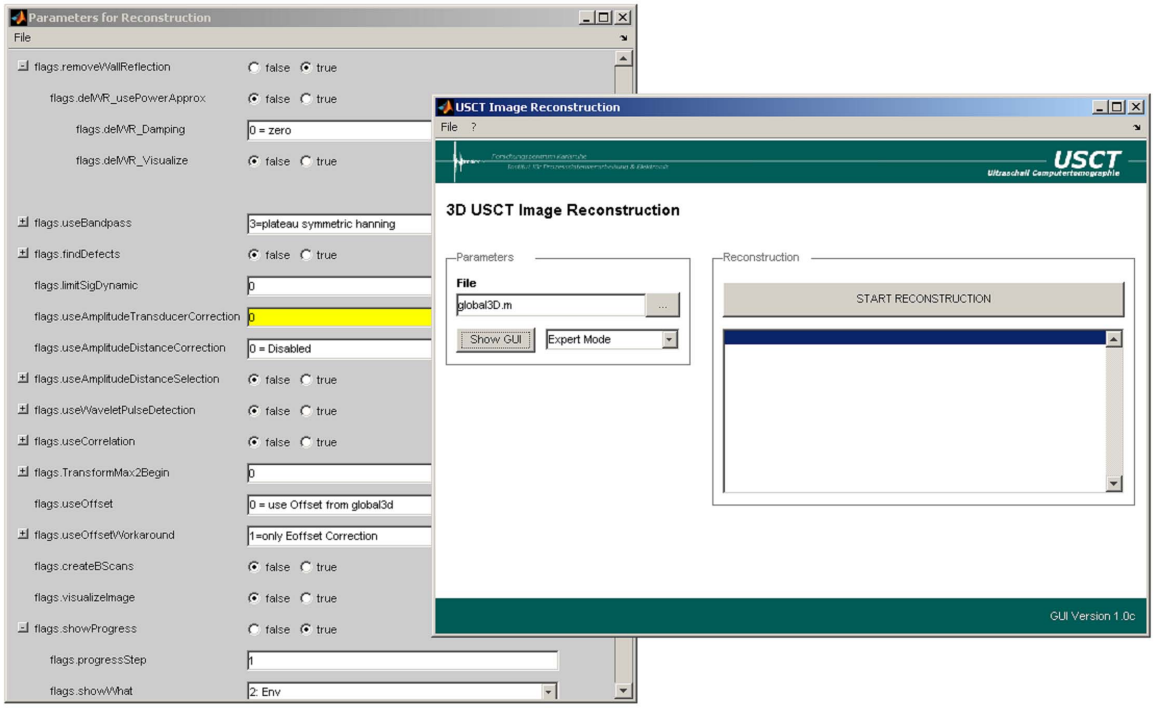


Figure 11. Dynamically generated GUI for changes in the parameter file of the reconstruction software.

and his capability level in software development. The second part investigates the appearance and functionality of the presented analysis software as well as the usability of specific functions. The third part of the questionnaire is based on the Software Usability Measurement Inventory (SUMI) [16]. The questions focus on the evaluation of efficiency, affect, helpfulness, control and learnability of the software. Until now, the questionnaire was completed by five scientists of the working group.

The results of the first part shows that all of the users in question develop software in MATLAB. They mainly create command line-based software tools, which they share with other scientists. The major part (four of five)of the users share the opinion, that training of students to get familiar with the USCT project is time-consuming. The results show that unexperienced users have problems to understand the coherency between A-scans, reconstructed images and the USCT aperture.

In the second part of the questionnaire, the users evaluated the user interface as clearly arranged. The users agreed on the simplicity of handling of the 3D model and selection of A-scans. They appreciated the partition into the three user interfaces, the presentation and browsing through 3D images and the extendability of the software. Another significant conclusion is the good applicability of the software for the training of students. Throughout all questions concerning the coherency of the A-scans, reconstructed images and the USCT aperture the users rated these function to be very beneficial.

The third part of the user evaluation was done using 50 questions following the SUMI questionnaire. For the analysis

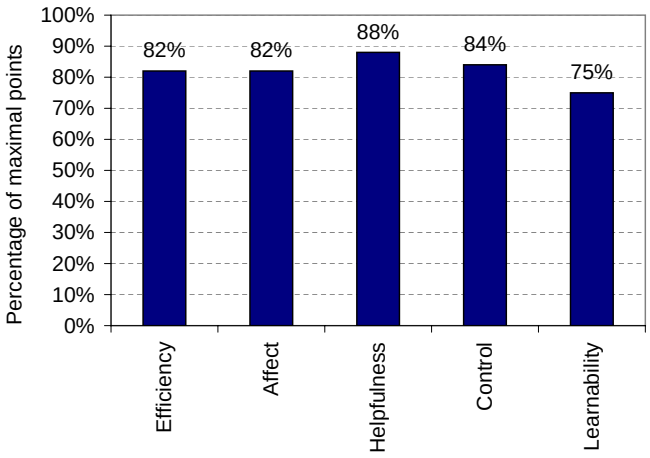


Figure 12. Results of the SUMI user evaluation questionnaire showing the attributes efficiency, affect, helpfulness, control and learnability.

of it, every question was assigned to one or more of the attributes efficiency, affect, helpfulness, control and learnability. The answers are rated with zero, one or two points, which are summed up for each attribute and related to the maximal number of points.

The overall averaged rating of the usability of the software is 81 % of the total number of points. The standard deviation is 12 %. Though the number of completed questionnaires is rather small, with a confidence coefficient of 0.95, the confidence interval can be acounted to $[65\,\%; 97\,\%]$, where the lowest value is still significantly higher than 50 %. This

encourages us to assume, that the developed software offers a good usability to the users.

Figure 12 shows the averaged result for the different attributes. The helpfulness of the software to process USCT problems was evaluated best (88 %), while there seem to be small difficulties in the learnability (75 %). Efficiency, affect and control are on the same level. Overall the results are on a quite satifying level of 75 % and up.

VI. Conclusion and Future Work

The described design and implementation shows the variability of MATLAB as programming language not only for numerical solutions but also for the design of complex software systems. With limited choice of interaction objects offered by MATLAB, a software system modeling the workflow of the USCT was created. The extendability is guaranteed by several interfaces leaving the programmer the choice of using already existing GUI elements or providing additional functions. By using parameter files whenever possible the system is adaptable to changes in the future – only these files have to be replaced.

As well as showing the capabilities of using MATLAB, the developed software also has a large benefit for the scientists. Before the software system was introduced to the USCT working group everyone implemented individual analysis tools for specific problems. The software now integrates most of these tools and provides a comprehensive GUI to access them instead of calling them by command line with multiple parameters. Together with the use of a subversion system everyone can now share analysis methods with the whole working group in a concise way. The integration of the reconstruction software was the next step to centralize all software systems used for the USCT in a comprehensive GUI.

The usability evaluation showed, that the software provides a good applicability for the training of students and new employees because of the new viewing method for the relationships between all parts of the USCT. The visualization makes the complex system more understandable than only reading documentation or even source code. By the integration of the reconstruction software the benefits of the analysis GUI can be ported to a second software used by the working group.

In the future the software will grow by adding new plugins and extending the basic functionality. Also the integration of the data acquisition software has to be improved. For the second generation of the USCT prototype the software can easily be adapted to the new transducer geometry by replacing the parameter files and revising the *A-scan GUI*. Furthermore the user evaluation will be extended to a higher number of attendants. The results of the user evaluation will be considered while extending and reengineering parts of the software.

References

[1] T. Hopp, G. Schwarzenberg, M. Zapf, and N. V. Ruiter. A MATLAB GUI for the Analysis and Exploration of Signal and Image Data of an Ultrasound Computer Tomograph. In *Proceedings of the First International Conference on Advances in Computer-Human Interaction 2008, ACHI 2008, IARIA*, pages 53–58. Published by IEEE Computer Society Press, 2008.

[2] H. Gemmeke and N.V. Ruiter. 3D Ultrasound Computer Tomography for Medical Imaging. *Nuclear Instruments & Methods in Physics Research*, 2:1057–1065, 2007.

[3] The MathWorks Inc. *MAT-File Format*. The MathWorks Inc., 2005.

[4] S.R. Doctor, T.E. Hall, and L.D. Reid. SAFT - the evolution of a signal processing technology for ultrasonic testing. *NDT International 19(3)*, pages 163–172, June 1986.

[5] M.D. Abramoff, P.J. Magelhaes, and S.J. Ram. Image Processing with ImageJ. *Biophotonics International*, 11(7):36–42, 2004.

[6] Volume Graphics Inc. Website. http://www.volumegraphics.com/, 2009.

[7] National Instruments Corporation. Website. http://www.ni.com/, 2009.

[8] C. Eggen, B. Howe, and B. Dushaw. A MATLAB GUI for ocean acoustic propagation. volume 3, pages 1415–1421, Oct. 2002.

[9] R. Stotzka, H. Widmann, T. Müller, and K. Schlote-Holubek. Prototype of a new 3D ultrasound computer tomography system: transducer design and data recording. In *SPIE's Internl. Symposium Medical Imaging 2004*, pages 70 – 79, 2004.

[10] M. Zapf, G. F. Schwarzenberg, and N. V. Ruiter. High throughput SAFT for an experimental USCT system as MATLAB implementation with use if SIMD CPU instructions. In Stephen A. McAleavey and Jan D'hooge, editors, *Medical Imaging 2008: Ultrasonic Imaging and Signal Processing*, volume 6920, page 692010. SPIE, 2008.

[11] G.F Schwarzenberg, M. Zapf, and N.V. Ruiter. Aperture Optimization for 3D Ultrasound Computer Tomography. In *Ultrasonics Symposium, 2007. IEEE*, pages 1820–1823, 2007.

[12] The MathWorks Inc. The MathWorks - MATLAB and Simulink for Technical Computing. Website, http://www.mathworks.com, 2007.

[13] The MathWorks Inc. *Creating Graphical User Interfaces - Version 6*. The MathWorks Inc., 2002.

[14] The MathWorks, Inc. Matlab file exchange: Scrollsubplot. Website, http://www.mathworks.com/matlabcentral/fileexchange/7730, 2008.

[15] The MathWorks, Inc. Matlab file exchange: Voxel. Website, http://www.mathworks.com/matlabcentral/fileexchange/3280, 2008.

[16] Drs. Erik P.W.M. van Veenendaal. Questionaire based usability testing. In *Conference Proceedings European Software Quality Week*, November 1998.

Using Secondary Information Sources to Generate and Augment Semantics of Design Information

Sascha Opletal Dieter Roller Steffen Rüger

Institute of Computer-Aided Product Development Systems, Universität Stuttgart
Stuttgart, Germany
email: {opletal, roller, rueger}@informatik.uni-stuttgart.de

Abstract

The reuse of design knowledge for use in CAD systems is a promising way to reduce time and cost during the design cycle. To support this, a semantic core for a novel type of informational infrastructure with the focus on supporting CAD systems is introduced, that allows to extract arbitrary subparts of the information base and use it efficiently in related projects.

The key problem addressed in this work is the automated setup and classification of information pieces within several involved knowledge domains. This is solved by connecting depending design methods strongly to information sources outside the actual CAD design environment with a focus on knowledge generation, distribution and application. As a result the approach will support problem solving within the geometric area through a system that can classify information based on context.

Keywords: CAD, Knowledge Bases, Semantic Relation

1. Introduction

Knowledge and information extraction, the aggregation and propagation of relevant pieces as well as relevant structures are needed in many application areas. As the overall goal is the efficient reuse of design knowledge, it is required to be able to extract arbitrary subparts of designs and to apply them to other design situations. Today this is only possible with form features, configuration models and other predefined methods, where all the knowledge and the procedures have already been thought of and have been integrated into the product models by a human designer.

The presented approach uses selected methods of different disciplines of information analysis to effectively support the setup and usage of the semantics involved with CAD models. These CAD model semantics have come a long way in the last years, but are still very low-level compared to the requirements of a system that allows easy reasoning beyond integrity checks. Though there are some abstractions that enhance semantics, like constraints and form features, the meaning of a certain part of a design remains in the dark, apart for special use cases, where all knowledge can be formulated [16]. If there is a special need to evaluate a certain aspect of a design, evaluations have to determine the meaning of a design through feature recognition or analysis of design graphs [15].

The involved techniques for a more generic approach to the problem are stemming from the field of computational linguistics, from where disciplines like information retrieval (IR) and information extraction (IE) have evolved. IR has the goal of finding information and evaluates the correctness or relevance of found documents or information sources. This is done by computing precision, recall and fallout (Eq. 1-3)[3], where R is the set of relevant documents, I the set of irrelevant documents, P the set of found documents, and N the set of not found documents.

$$Recall = \frac{|R \cap P|}{|R|} \quad (1)$$

$$Fallout = \frac{|I \cap P|}{|I|} \quad (2)$$

$$Precision = \frac{|R \cap P|}{|P|} \quad (3)$$

Recall then defines the found amount of documents, precision defines the amount of found correct documents and fallout the amount of not found relevant documents. A problem with this approach is that all correct documents have to be known in advance to optimize the knowledge search and to evaluate it using recall and precision. Once a set of useful documents is found, IE can be used to focus on the gathering of meaning out of the information sources.

[5] defines this as ”An IE system takes as input a text and ’summarizes’ the text with respect to a prespecified topic or domain of interest”. This works generally on unstructured text within a fixed domain that defines how to handle and combine the found information. This process is typically even more

constrained and only works on given scenarios, which can be described as the focus within the work domain. As an example, the domain could be 'Economic news' and a possible scenario for it could be 'Changes in Management Positions'.

A well defined scenario [28] allows that techniques like stop-word elimination, filtering of known subjects and reasoning can be used to strip unimportant expletives that are irrelevant for the task. A more general approach, but also focused on domains, is the field of Knowledge Discovery in Databases (KDD), or data mining. It has the goal to recognize unknown patterns and relations that are not specifically encoded or modeled by using statistical analysis of data sets. An example is to find changes in the consuming habits of customers to evaluate products or to rearrange a business strategy. Data mining is subdivided into different disciplines, motivated by the processing methods through which the data is passed. The focus on **data description** is to give a compact representation that is reduced to the essential information. The **differential analysis** tries to identify data sets that are deviating from given norms or standards. The **dependency analysis** tries to make out relationships between the attributes of information objects. **Clustering** is used to segment the data set into groups of interest.

While being focused on special domains or special purposes and scenarios, a multi-domain approach has the the advantage of being more practical in heterogeneous work environments as found in companies.

2. Knowledge domain and Aggregation

In the product design process, CAD systems cover the geometrical design aspects. PDM (Product Data Management)/PLM (Product Life cycle Management) systems store documentation. CRM (Customer Relationship Management), SCM (Supply Chain Management) and ERP (Enterprise Resource Planning) systems are used for accounting, materials administration, NC programs, or information for advertising. The overall knowledge process depends not only on explicitly structured and stored information pieces but also on unstructured sources like email that is exchanged between coworkers or even information that is not explicitly modeled and stored in the system. The aspect of explicit and tacit knowledge strongly influences the work processes [14][20].

In order to input data into a knowledge base and to be able to reason on it, the construction framework has to i) consist of rules how to build it and ii) has to have as a base a specified domain. There are lots of frameworks for i), like ontologies, semantic networks and so on, but ii) is still a major problem when it is set up from a multitude of domains as described above.

The goal is to avoid the case where somebody needs to write down which concepts belong to the domain of the application and which not - or even model the concepts first. One reason for this is, that everybody has different concepts in mind and also if two people think about the same piece of equipment, they could use many different names for it. [17] describes this by introducing the meaning triangle to relate symbolic descriptors to objects. Checking for correct descriptive symbolics is not the focus of this work, but to correlate given symbolics correctly to each other. This also covers situations where multiple symbols for an object exist, like different names or a filing with different numbers.

Generally a knowledge modeling framework is a specification of a conceptualization. This is a formal description of artifacts and their relations, that are used to build a common base for the concept formation of a person or a group of persons. The goal is to describe the concepts of human thought and communication in an unambiguous way by using a formal fixation of concept hierarchies, relations and involved attributes. This allows the use of computational tools for inference, extraction of information and the generation of searchable indexes. The main approaches to represent knowledge are either symbolic or connectionistic. Symbolic knowledge is formulated in schemata or rules of the different expressions of a model. A connectionistic system stores the knowledge by training and reveals the knowledge through interaction, which means that there is no way to access knowledge directly since the storage process is not transparent.

A knowledge representation can be generally classified by correctness, power of expression, efficiency and complexity. The content of one representation can usually be transformed to another representation, but there is a risk of semantic loss, depending on the power of expression of the target representation or equally a source representation that does not deliver the required input. The chosen representation should be connectable to all involved information resources and will act as an instance that mediates and aggregates the information.

Three frequently used formalisms to model knowledge are semantic networks, frames and ontologies. Generally they consist of concepts and relationships between concepts. A concept can be an entity of many types ranging from structured frames to unstructured data files. The relations (comparable to ontologies) usually have a base semantic similar to the UML, but are often extensible to arbitrary semantics. Expression is high and versatile, but structures have to be built manually according to a meta-level. A designer is required to define the base structure and uses elementary methods to set up concept structures and relations. Base semantics of relations are usually:

- IS-A, A-KIND-OF (a kind of) used to represent heritage or information of generalization
- PART-OF, HAS-A-PART used to represent aggregated information
- MEMBER-OF, INSTANCE-OF used to represent instantiation and individualization

The product development is usually carried out within a knowledge frame that spans between persons and teams. There exist relations of various semantics between all involved (also abstract) units like teams, db-objects, prototypes, projects, and so on. Though some relations are not explicitly encoded, they are supplemented by a persons mind while working with the given data set, which means that the collection of data together with implicit knowledge forms a semantically connected knowledge system.

Applied in product development, the vertices of such a system represent data objects and the edges represent the semantic relations between the objects. Once the structures and relations are set up, they are fixed. The user of the system can only generate instances by using the structures of the network to fill in data. Most knowledge representations are tailored to specific use cases, but can also be enhanced or modified to be used in other contexts. However, structure and expression are often too fixed, while rules are tailored for specific reasoning.

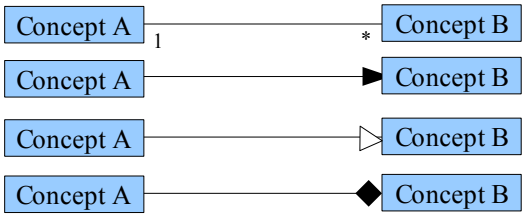


Figure 1. UML relations between classes

Similar semantics are used also by the wide known UML notation (see fig. 1). From top to bottom is shown the graphical annotation of an association with multiplicity, the directed association, the generalization, the aggregation or composition.

Derived from such semantics, the knowledge propagation to people and places where it is needed can be decided. This will save huge amounts of time, but needs a powerful annotation scheme. Changes in a CAD model can for example trigger actions in depending data like cost calculation, manufacturing processes and so on. Such a system needs an active component to be able to connect knowledge, push knowledge or coordinate knowledge generation.

A combination of different knowledge acquisition methods and representations can be used to form a system that organizes information without the need to form structures with fixed concept frames for the deposition of knowledge. The information should give rise to structures and not vice versa.

3. Related Work

Recent approaches to model knowledge often use static setups of rule bases and concepts to capture the knowledge domain. Currently used [25] informational infrastructure for the development and manufacturing of products distributes different information types over specialized systems.

A system with an intelligent tutoring agent is described in [26], where the preprogrammed domain together with the allowed actions form the base for reasoning. [10] suggests a domain independent knowledge manager that is very flexible in terms of application, but omits how the domain knowledge can be gathered and be distinguished from non-domain knowledge. In [22], the domain knowledge is acquired automatically by evolutionary learning but on a very limited domain. Traditionally the search for domain knowledge stems from NLP (Natural Language Processing), that searches for meaning in unstructured free text, based on large text corpora of a specified domain. A method to use meta-information to classify documents based on the citation information of authors is presented in [9].

A domain independent approach is described in [21], but the topics and domains have been preselected, which is undesirable for the problem addressed in this paper, where more than one domain is involved. [12] uses a highly sophisticated annotation to classify the physical effects of single mechanical modules. Using them as building blocks, mechanical functions can be created by connecting the available and desired input and output forces. [9] adresses the reasoning aspects behind constructability.

A more interactive approach that uses evolutionary algorithms to design products is presented by [6]. The validation function from each generation to the next is the human who tells if he likes a generated design or not, which resembles an information retrieval.

Many more approaches exist to solve domain specific problems. The problem discussed in this work is in contrast to these approaches, as the focus is to establish a framework that is able to bootstrap itself and store knowledge domain independently.

4. Capturing Design Knowledge

The aspects involved in developing an object or structure to meet set criteria are considered as design knowledge. This includes the lead to a solution, the process how criteria of the application area are met, and also the solution itself. Many aspects of the geometry fall into this definition; some can be analyzed automatically, others can only be captured by hand. Generally an algorithm can log the use of utilities and the steps of creation for a specific object, but the intention of the designer and what the object should be

good for or where it will be used in and what else would be appropriate in this specific situation is beyond the perception of the computer. There are design goals and supporting information that is unique to special domains, but also special domains are involved within a project, that don't cover mechanical assemblies and require special knowledge capturing methods as they also contribute to a solution.

The acquisition and storage of the different types of domain knowledge can be characterized into domain independent and domain dependent knowledge. Domain independent knowledge should be equal in use and acquisition across design domains. This can be captured at a higher abstraction level [19] above the feature based design and the product configuration. Form features represent geometric macros that capture and preserve the design intent as geometric setups and machining instructions. They can be considered domain independent, since they are used to model geometry in many projects. Certain groups of semantics which are annotated or classified through usage are similar across the domains and can therefore be directly transferred. The basic geometric construction methodology is domain independent and is used in every design system. Next to the geometric information there is process information within the CIM/CIE environment that has influence on the design process. Those organizational aspects can be modeled in the same way across domain boundaries.

Domain specific knowledge in contrast is captured by information and methods that are unique to a design domain and cannot be transferred across domains. As example consider the design of a molding form that is used to form components out of mold. In order to get the ready component out of its form, the form needs to be fitted with ejection channels, where the component can be pushed out. The placement of those ejection channels needs to be formalized and calculated by a specialized tool for each new molding form. Such knowledge is domain dependent, since it only applies to the domain of molding forms.

It is however most certain that this knowledge can be reused within different projects that are developed within this design domain. Those domains need customized treatment for capturing, transfer and application of reusable knowledge. The general approach to handle these problems is very similar. The applied construction methods can be generalized and be fed with specific knowledge for the actual domain. These specialized methods can be seamlessly integrated into the information process through a standardized interface.

The feature recognition serves as a first measure to compare the design situation at hand with the stored information in the knowledge base. Features are used in conjunction with restrictions to capture the design intent in a product model, so that changes do not destroy the original intent of the designer. Features of a design can be captured in two ways: During the actual design process, the designer assigns features to his geometric structure knowingly by using a certain feature function and unknowingly by creating a structure that holds certain features that were not explicitly constructed. The features that are assigned through feature functions can be directly annotated in the database and be refined by questioning the user to input his intent. Implicit features and unannotated features can be found during a preparation phase of the knowledge base, where new information is being gathered out of the stored objects. This includes a feature recognition process to find additional knowledge about objects through the presence of features.

The feature recognition is also very important in the information retrieval process where a design problem is analyzed into a new feature structure and this feature structure is then searched in the knowledge base as an exemplified query. Additional information is supplied through the characterization of the design situations in the overall information process. For every applicable situation found through the recognition of features, there has to be a measure to decide how much of the found information is applicable to the given situation. This is done by a similarity function that takes into account the geometric similarity and the process similarity. The geometric similarity is computed through the involved objects in a design situation with respect to the needed solution. The similarity of objects can be determined through design history, annotation and features. The similarity of the stored design situations is compared through the similarity of the involved objects to the objects in the current situation. Further comparison is done on the information process level, such as context in terms of design teams, as well as requirements and input from different organizational units of the company.

5. Modeling Information

The conventional approach to knowledge based design means that the user fills the knowledge base with rules and by doing so decides how the product model will behave or can be reused [2]. Knowledge in this context can be described as [14]: "Knowledge is information that is relevant, actionable, and at least partially based on experience."

A special focus has to be on the starting phases of the product development process as these phase are characterized through a high need of interdisciplinary information. A knowledge base that is used in such an environment needs the capability to cover objects and relations of many different kinds. These requirements emerge from the dynamics of the product development process and as a result, the knowledge base does not

represent a stable and consistent state, but needs to reflect several development processes that are running in parallel. The most important tasks are to find and use methods to synchronize the work processes and to be able to recover from inconsistencies. The methods of representation for the product data have to support incremental refinement and extensions to the knowledge which is gathered during the product development process.

The process of knowledge acquisition is usually guided through a descriptive framework that serves as a meta-level to set up the rules and methods to fill in data (see fig. 2).

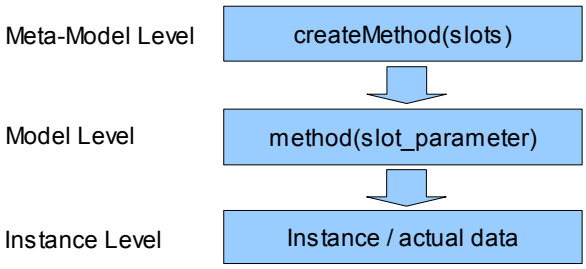


Figure 2. Levels to build a data model

A good knowledge representation and the associated tools should support the need to acquire, consolidate, and distribute information among involved clients to create a measurable advantage. The collaborative environment should support this through a collection of processes and tools, which will lead to new ideas, processes and techniques. Several knowledge modeling techniques were developed over the years:

- Frames
- Ontologies
- Semantic networks

The individual models each focus on different aspects of knowledge. Ontologies as an example are suited to provide and organize additional CAD process information in an effective way, since the development of a product is done within a project organization and driven by persons and teams, which are themselves considered as acting objects in the information structure. They create several different work objects, such as requirements, descriptions, CAD-models, NC-programs, or material for advertising. Between these logical units exist relations of different kinds.

Those ingredients define a semantic network. An extension, called the *active* semantic network (ASN) [24][7], has been developed at the Institute of Computer-aided Product Development Systems at the Universität Stuttgart to support active propagation of knowledge. It provides means to store all information that is created during a cooperative development process. The main component is a mechanism that executes automatic actions, driven by the states and situations during the use of the semantic network, which is of great importance to coordinate the development in an effective way. The structure of the active semantic network consists of objects that are connected in a form that resembles a net. In this net, the vertices represent objects of the product development knowledge and the edges represent the context and relations between those objects. All relevant information for the product development can be stored in this active semantic network, such as input from quality control, marketing, buying department, service and recycling. The knowledge model of the active semantic network is dynamic and can support the growth of knowledge and new knowledge types during the product development process.

The active part of the semantic network allows for the propagation of changes through the whole net and the triggering of actions based on those changes. It bears the following functionality:

- Inference mechanism
- Message passing
- Communication handling
- Execution of tasks

Inferences are used on the knowledge base to execute rule-based recombinations of information and automatic calculation of formulas. Through the passing of messages to users or the establishment of communication channels, the system can react on problems that it or the users cannot solve by themselves or where human attention is needed. A novel aspect when compared to other knowledge representations is that the active component fills the role of an assistant to the designer. It can execute routine tasks and bring the attention to problems that have occurred during the development process.

The active semantic network can support the designer with knowledge from an expert that would be out of reach or hard to come by in a standard work environment. The designer can be notified if a design decision has consequences on the subsequent product development phases, such as the manufacturing process and product cost. By providing the knowledge of areas where the designer is no expert in, the active semantic network takes the place of a colleague with the necessary expertise.

The problem to define the domain of knowledge as a basis for the subsequent usage in reasoning processes is not easy. It is often done by hand and thus tailored explicitly to a specific application area. However domains and contexts in all industries and application areas differ a lot, even between companies within the same field of activity. Thus, the automated set-up and maintenance with respect to new requirements is an enormous task. A knowledge base that builds itself and also has the ability to self-organize is the key for a substantial push in the re-use of a company's stored information where it is desired to keep the integrity of

the structures of databases. This motivates the approach of a layer on top of all involved information systems, that organizes and groups information pieces according to their relation and relevance for projects.

6. Use Case: The Information Need for Sheet Metal Design

The application and data flow of the semantic core is described from the perspective of a CAD designer that interacts with the knowledge system. Many factors drive the informational need with cost being the most significant, as the work area of development and design contributes 75 percent of the total costs.

The capability and experience of a human designer generally makes or breaks a project in terms of initial quality and the time to reach a satisfactory base construction. As the initial design is taken and refined in several iterations [23], using input from other involved teams (quality control, mechanical testing, production units, …), it is desirable if the system could automatically provide the information from established solutions and paid attention that the product design meets the requirements (norms, cost, quality …).

The design process of a new product requires precise knowledge on function and manufacturing of the final product, as the function of the product is determined by its mechanical construction. The quality of the construction depends on how it was manufactured and which materials were used. Also most new products are not designed from scratch, but built on or advance an established base construction of previous designed and proven products.

The designer has to take all these requirements into account when starting the design process. The goal is to integrate human design knowledge into the CAD system, so that certain design decisions can be made automatically or be suggested in the relevant context to the human designer. A "semantic CAD system" can be formed, based on a semantic core, that has the primary task to support the design process by operating as an assistant to speed up the process and guarantee high quality standards by supporting the design process as detailed as possible. These supporting steps consist of:

1. Idea of the design / 2. Articulation by entering it into a CAD system / 3. Proposal of the system / 4. Transfer or modification of the proposed design

A precisely articulated idea, aided by the support of adequate design methods will make it possible to give the designer suitable help that is tailored to his needs. To see the information flow during the CAD process, standard problems will now be discussed to motivate the presented approach and to show the benefits it provides. The proposal of the system is derived from the consideration of the knowledge base, which will be checked according to three different controlling areas: design controlling, component controlling and realization controlling, with each of the controlling areas being presented below. The necessary data flow for the monitoring is made up of several levels:

1st level: design work
At this level all information about the user interactions, the used materials and the geometry of the design is collected which represents the raw data for the knowledge processor.

2nd level: knowledge processor
The knowledge processor receives the raw data and generates knowledge by consulting relevant sources of information that are from the same context as the current workpiece or work context. After aggregating relevant information, the knowledge processor synthesizes a helping information piece and presents it to the designer or applies a modification directly to the work-piece.

3rd level: decision of the designer
The designer has several options how to continue with the provided information. Proposed design methods of the knowledge processor can be applied and change the design. Alternatively, the information on the current design only appears passively, is noticed and the design work continues. Feedback is gathered by other means, for example when requests are made for more information on the same topic. The decision will be monitored and used to evaluate and refine the decision score.

The global problem is to recognize the intentions of design work in the correct context within the information process and stored information. The knowledge processor is responsible to gather all the information needed to support the design functions. Therefore all relevant data has to be extracted from a modeling process and also from the other information systems involved. This is guarded by three different surveillance types:

1st section: design controlling

- Verification of design rules for the design, manufacturing, installation and cost reduction: rules are extracted from active and older, already completed projects. With these rules a learning process is able to use relevant and correct older structures for new designs with the goal to minimize design errors
- Connectivity of the knowledge processor to a quality system: in case of defects or frequent customer complaints their cause has to be analysed. If the design is responsible, it is recorded for future projects to avoid mistakes
- Compliance with the requirements regarding existing tools and machinery: avoid expensive

acquisitions of special tools if it is possible to manufacture with existing work equipment

2nd section: component controlling

- Comparison of similar designs: a work history is stored and accessed during the phase of modeling. In the case of similarities between older and current designs, proposals are made by the system. If the designer agrees with the proposals they are automatically included in the current design
- Integration of products from third party manufacturers and completed parts: the use of standard parts has a high potential for cost reduction. Therefore parts will be identified during the design as third party parts that can be bought cheaply and thus avoid the redesign of parts or unproven designs

3rd section: realization controlling

- Recognition of failure in design: not feasible designs should be detected as early as possible so that the designer does not waste time in producing something incorrect
- Consideration of the material properties: if a bending needs to be made in the sheet metal module and the sheet metal is too thin, it has to be avoided by giving a corrective proposal
- Recognition of the possible unfolding of sheet metal design: invalid designs have to be avoided

As example the following design situation is considered where the designer needs to be advised why it has a bad shape. An optimized shape is provided by the knowledge processor with support of the intelligent CAD methods based on the controlling sections.

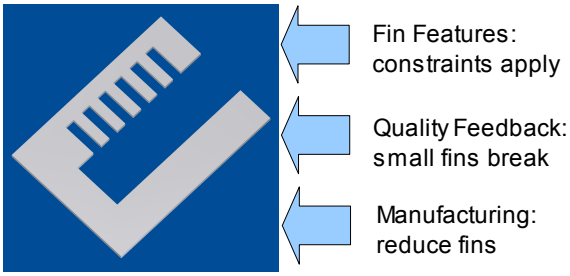


Figure 3. Sheet metal with thin fin elements

The model is checked for various criteria. On one hand it is checked what kind of material is used and what material properties it has. On the other hand the shape and form of the model is analyzed. In this example the material used is a metal plate (see fig. 3). The fins were created with a feature that was used six times. A feature consists of one or more modeling elements. The analysis of the design recognizes that there are multiple (five) fins standing out, which are relatively thin developed. The problem in this design is that too fine structures are not in accordance with the design rules derived from manufacturing machinery constraints and quality feedback. In the production a fine stamp would be required, which is very sensitive to destruction. It is also very likely that too subtle fins will break off in the process of manufacturing.

The design has to be examined whether the fins are not too close to the edge of the metal sheet or if it might be too thin to keep the fins in a solid position. To modify this model into a better shape the slim feature is modified in the way that two of the slim features side by side create a new thick feature. Then the system removes all slim features from the model (see fig. 4) and by doing this reuses derived and proven information.

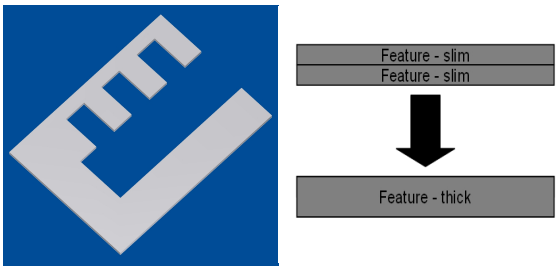


Figure 4. Optimized sheet metal fins

The thick feature will be applied three times to the model. With the thick feature two massive fins are generated, which are much less sensitive to damage in the manufacturing process. A robust stamp can now be used with more resistance against break off. The spacing of the fins to the sheet metal edge was classified as uncritical and was retained.

The required knowledge has to be stored and retrieved from the knowledge base to be applied to a relevant design situation. This means that procedural and other knowledge has to be classified and related to the current and possible future working contexts. The knowledge to address the problems described above could be formulated as:

```
begin compare_features;
  if number of features in model browser is >= 2 then
    count all features of equal form;
  else no modification possible;
  end if;
end compare_features;
begin merge_features;
  if count of equal features is odd number then
    merge = true;
    create new features;
  /* 2 slim features merge to 1 thick feature */
  else merge = false;
  end if;
end merge_features;
begin start_modification;
  if merge = true then
    save positions of slim features;
    calculate new positions for thick features;
    delete slim features from model browser;
    insert thick features at new positions;
    msgbox with information about modifications;
   end if;
end start_modification;
```

These and similar code snippets can then be classified using the keywords „features“, „constraints“, „modification“, „manufacturing“, „sheet metal“, and „quality control”. The use of relevant knowledge has to be decided from the working context, that is partly derived from the user interactions and partly of the design situation and previous knowledge that has be used and applied. The situation could be as described in fig. 5, where the relations between the areas are given. The vertices then represent the actual knowledge to be applied.

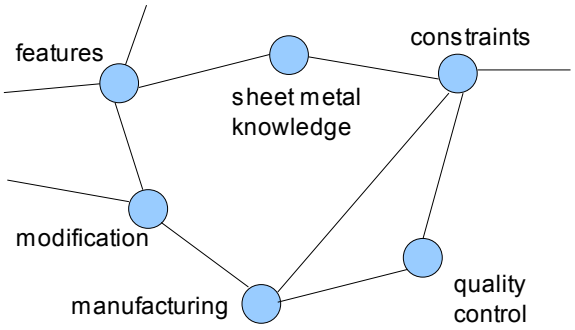


Figure 5. Knowledge for sheet metal design

If the user works with features and executes a verification process for manufacturing constraints or quality control, the system guides him to the sheet metal knowledge and applies related information accordingly. A score of quality through evaluation of several sources linked to relevant topics then forms the base of the semantic core. A great difficulty is to store the geometric information in a readily available form for the adaptation to design tasks. It is impossible to extract design knowledge out of a pure geometric design. The machine would not know what the functions and intent of the single product parts are.

To know which stored products could be a knowledge source for a design process, the products have to be semantically annotated. In a modern CAD system this is already done in a very limited way. If a feature-based system is used, then the product model stores the type of features that the engineer uses during the design work. This can be driven further, in a way that the designer gets a semantic toolbar and annotates product parts and groups by hand while he constructs them. From this annotation the system can learn and through comparison of geometric elements it can annotate other elements in the database using a probability function.

The annotation layer is needed to be sure, that the topic of the analyzed design situation fits to the current design problem. If the analysis was only based on geometric properties, then there would be many unwanted solutions. Almost any geometric shape can be transformed to fit in a given situation, but it has to be decided which geometry is wanted and which is not useful in a given situation.

7. Building a Semantic Core

Various information sources need to be integrated into the knowledge process as discussed in the design example. Although the information flow is directed to the requirements of the designer, all other involved knowledge processes can profit in a similar way. The approach superimposes the existing information sources with a guiding structure to provide a new level of insight by providing more contextual information than it is possible with the isolated systems. The need for reasoning and knowledge aggregation is therefore greatly emphasized.

One key feature as described in the above sections is to avoid the need to build up structures or to identify, limit or input the domain by hand. Therefore the first step is a classification of the involved information systems that are used as resources provided by the company. This will be the foundation where the information is stored, identified and connected to relevant information from other systems.

Some systems may have a dominating role regarding the intended purpose, e.g. an information in a PDM system would be regarded as a primary source, but information extracted from email would be treated as a (maybe unproven) secondary source that can enhance the value of the primary source. The following step is to set up an initial key set as a seed from which to build the base core semantics. By extracting significant descriptors like document or project names and relating those keys, the information resources are analyzed for collocations of pairs of those keys. To refine the confidence of a found or supposed relation, as shown in fig 6., the internal base is switched with external resources like Wikipedia [27], Google [11], or ontologies like OpenCyc [18]. Those resources cover many domains and are used to identify relationships between the key identifiers. The results help to compute if there is a strong or a weak relationship and if there are more relevant key identifiers that were not found during the initial seed.

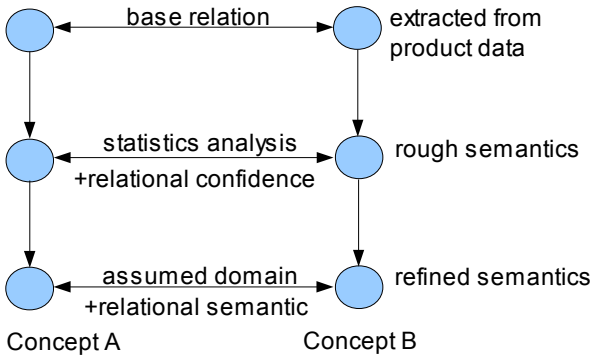


Figure 6. Finding and refining semantics

Through the extraction of internal and after that again external relationships of concepts by methods of

information retrieval, the semantic confidence is rising. The semantic relations of interest from the aspect of reasoning in CAD are relations of the type:

"part of"/ "works on" / "similar to" / "instance of" / "new version of" / "has relation to"

This seed acts as a first rough information structure which is not static but refined in subsequent steps, with the work flow and relations of the applied methods shown in fig. 6.

The semantic core drives all informational processes regarding retrieval and push of information. A reference is built by classification of all database content for fast access. Besides the overall classification of knowledge resource, there is also a need to establish an internal resource semantic. The goal is to be able to extract relevant knowledge for active knowledge delivery based on context. To keep the semantic core up to date and valid, there is a need for constant training on the proposed model, where a significance evaluation of the pushed content is done by observation how the knowledge is used in criteria like access, context and expanding of usage paths along the network.

A data set that would lead to the situation described in step 6 of fig. 7 would look as shown in table 1 with knowledge sources α,β and γ, where the influence strength is $\alpha>\beta>\gamma$, given by the type of the resource.

Table 1. Relational Information

	A	B	C	D	E
A		αβγ			α
B	αβγ			αγ	αβ
C					α
D		αβγ			
E	αγ	αβ	α		

Table 1 shows the results delivered by the information resources as relations between the concepts A-E. Raising confidence, apart from the influence strength of resources and the amount of congruent found relations, can be done by monitoring the usage of the semantic core. This is used as an approach to avoid asking the users for feedback. After the initial setup, the relations can be further refined through usage observation based on criteria like access, context and time of usage or informational movements along the network. This is a passive process, since it is difficult to detect if the piece of knowledge is directly copied or used and modified. A heuristic that enables the system to cope with contradictions and user disagreement is introduced at the end of this section.

Problem solving capabilities can be integrated at this point through inference mechanisms and also external solutions can be requested automatically by sending messages to experts. If someone introduces a new concept or term into the databases by creating new resources or documents with a key that was not in the original seed, it is evaluated against the original seed and integrated into it. If the creation happens within a given context in the semantic core, e.g. while working with a certain resource, the relations can be applied to the newly introduced component automatically. The final semantic core takes as input the relations between subcomponents and the overall design tree. Fig. 7 shows the process of setting up relations and the subsequent building of advanced relational semantics.

After the initial identification of concepts, they are statistically verified by using publicly available services and information resources like Google [11], Wikipedia [27], forums, product databases or web crawlers.

Identifying standing terms

Standing terms are composed of smaller units, but form a key term when combined. They are easy to identify. A search for standing terms is done by exact string match and delivers the following hit values:

sheet metal 10.2m deep drawing 8.6m

Singling out random terms

Equally, random terms with no meaning or no relevant meaning can be singled out or prepared for deep analysis by taking apart the components. For random terms or a non-standing term like "fin drawing" very low hit counts are returned:

Sheet metal xcad1	1 hit
Sheet metal cartest	70 hits
fin drawing	269 hits

Identifying and separating domains

The above approach can also be used to identify domains by searching for strongly connected concepts and dividing them from other hot spots. Since we use a global domain of all domains, there can only be a tendency of strong belonging computed, but not the final truth, since all domains are interconnected at some point without a static border separating them.

Table 2. Domain relations

	metal	extrusion	plastic	design
metal	-	3,8m	6,2m	13,9m
extrus.	263k	-	275k	4,1m
plastic	6,4m	261k	-	8,6m
design	25,6m	4,1m	8,7m	-

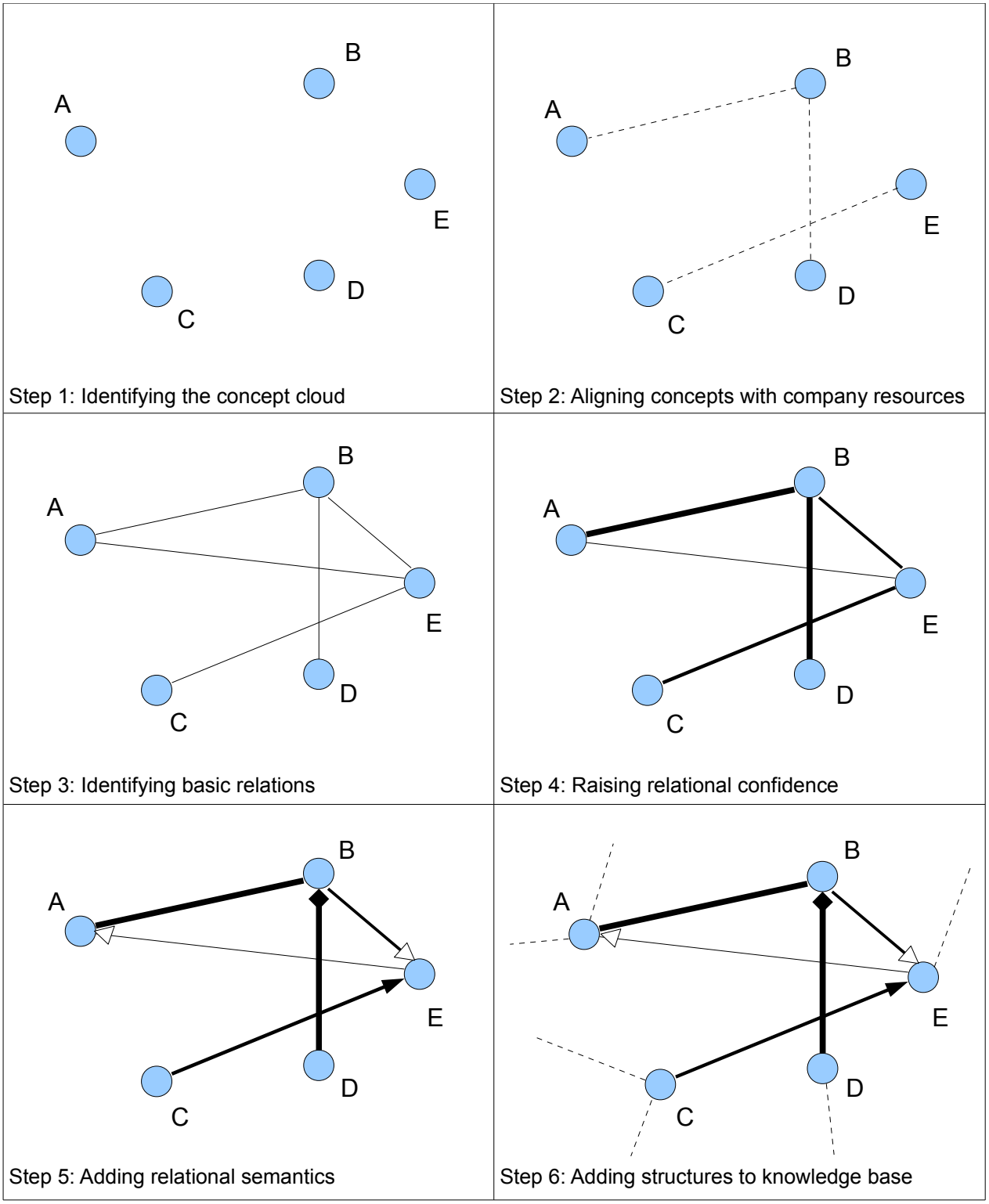


Figure 7. Setting up relation semantics between concepts and adding confidence

Domain relations

Combinations of terms reveal if they are semantically connected or not. A few examples of hit values by Google are given in table 2. It is interesting to note that different sequences of terms result in different hit values. The values can be cross checked with the inverse values, for example the combination of the terms “extrusion” and “plastic” delivers around 300k hits, while subtractive hit values are returned as:

extrusion -plastic 5,7m hits of 9,4m hits (extrusion)
plastic -extrusion 209m hits of 271m hits (plastic)

Domain division
Similarly as identifying the domain membership of a concept, the division and intersection points of domains can be computed (see table 3).

Table 3. Domain division and intersection

	metal	extrusion	plastic	design
crop	743k	115k	691k	1,2m
wheat	674k	172k	597k	741k
tractor	247k	126k	772k	925k
field	10,2m	3m	5,1m	41,8m

Reverse values confirm the findings of divided domains: metal -crop 54,3m / 619m total
metal -tractor 52,7m/619m total

Also intersections are visible since the term "field" is too general and appears in many domains. This is shown through high hit values of combinations within the domain of sheet metal working, but also within the domain of agriculture:

field crop 3,2m / field tractor 2,1m

The results strongly depend on the distribution of domain content in the search databases. The rough belonging to domains is computed as the sum of both ordered search results with a cutoff that filters random hits as: order_normal + order_negated - 2*cutoff > 0, where cutoff is empirical at around 900k and has to be related to the average distribution of positive hits.

Refined semantics
Based on the computed rough semantics from domain and non-domain hits, a dedicated domain source is added to provide higher semantic confidence between the recognized terms, using reliable information sources like encyclopedias or ontologies.

A list of paragraphs of the Wikipedia entry on „sheet metal" [27] is shown in fig. 8 with a section containing a classification of the context in fabrication and similar topics regarding metal working.

Contents
1 Processes
1.1 Stretching
1.2 Drawing
1.2.1 Deep drawing
1.3 Cutting
1.4 Bending and flanging
1.5 Punching and shearing
1.6 Spinning
1.7 Press brake forming
1.8 Roll forming
1.9 Rolling

Figure 8. Paragraphs about "sheet metal" [27]

The Wikipedia entry on „sheet metal" describes the relations to other topics:

"**Sheet metal** is simply **metal** formed into thin and flat pieces. It is one of the fundamental forms used in **metalworking**, and can be cut and bent into a variety of different shapes. [...] extremely thin pieces of sheet metal would be considered to be **foil** or **leaf**, and pieces thicker than 1/4 inch or a centimeter can be considered **plate**."

Combined with the statistical information of the semantic relations and the involved terms, the concepts can be extracted. A simplified part of the extracted semantic graph for sheet metal and its surroundings is shown in fig. 9.

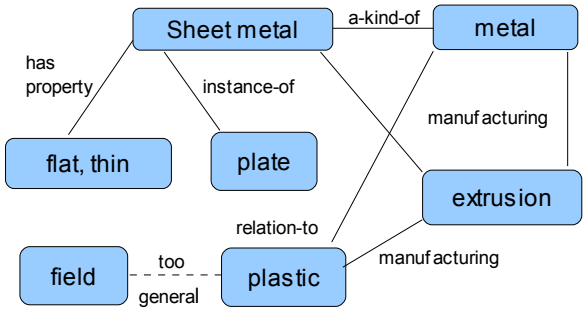


Figure 9. Semantic graph for sheet metal

8. Comments on Stability and Regression

The need of a semantic core is visible in most efforts to build an advanced CAD system. The requirement to build the domain by hand is usually met shortly after finalizing the building framework. The approach described in this paper can be applied to all domains where enough data exists to extrapolate domain dependencies and semantics of relationships between concepts. With regard to the stability of the discussed examples, the most important aspects are:

***Reliability**:* the approach to use unfiltered web-content to compute concept distribution has the advantage to be able to use multi-domain information. The drawback is, that the correctness cannot be guaranteed, since all domains are mixed into a whole. Identified hot spots can be searched for in reliable sources of domain knowledge like encyclopedias. The problem is, that not all information has to be in there, which is in turn solved by the general multi resource approach, that considers multi domain information, which is supposed to be correct within the domain itself and then extended beyond domain borders.

***Knowledge Acquisition**:* speed and reliability may be increased by using an ontology as the starting point and as a base to guide knowledge acquisition when building knowledge based systems. Ontologies have however the disadvantage that they are often fixed to a specific domain or even subdomain which makes it difficult to use them for a multi domain interactions.

Specification: if company resources are matched against the external information sources and used as a filter, then the domain knowledge of a company can be extracted as a specification for all involved information which can then be classified accordingly, so that the established structures of the company serve as an additional source of how to relate the information.

Re-Usability: the knowledge base is the foundation for a formal encoding of the important entities, attributes, processes and their relationships in the domain of interest. This formal representation supports the reuse of information and shared components in subsequent projects.

Search: the structure of the knowledge base may be used as meta-data, serving as an index for a repository of information, delivering meta-data for facilitating searches for product knowledge.

Maintenance and regression: the presented approach is able to reevaluate the content over time. This eliminates the need to maintain the system as it keeps itself up to date automatically and eliminates weak semantic links. By using a weighted voting algorithm to train and evaluate the confidence of semantic connections, a simple heuristic takes into account the interactions of the knowledge sources on the target relation. Relations found and verified in information resources place a weight on the relation, based on their relevance, as shown in table 4. Since the users have to use the knowledge base, they also have impact by voting (see table 5) to influence and reject structures. By using three areas of reached weight (see fig. 10), it is decided whether a relation is stable, discarded or still in the evaluation phase.

Table 4. Voting weights for sources

Source	Found	Not Found	Contrary found
Google	1	3	2
Wikipedia	2	2	2
OpenCyc	3	1	2

Table 5. Voting weights for users

	Agrees	Rejects	New Relation
Using	1	3	5
Browsing	2	2	5
Regression	3	1	5

This means that changing requirements can be automatically considered by the design methods based on the semantic core. When new company rules or norms become effective, the design system tells the designer where he is outdated.

Designs depending on machining aspects can be verified by linking the designs in the knowledge base and have it analyzed for dependencies. Transfer of ideas between designers is actively promoted since they all use the same knowledge base and are exposed to influences from their colleagues. Although all the above aspects can be applied for many design systems, the emphasis on a boot strapping setup for a multi domain knowledge base is a unique approach.

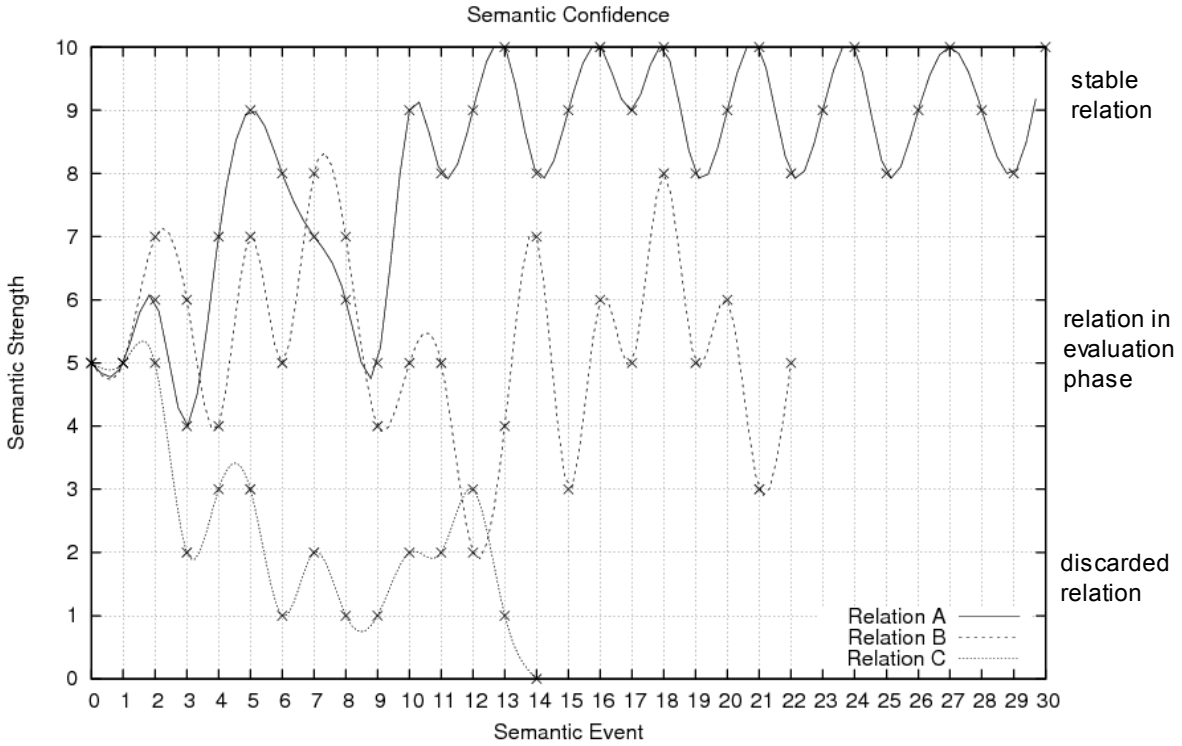


Figure 10. Phases of stable, evaluated and discarded relations

For a given context, the knowledge base supports content such as product models and geometry relations as well as company related standards, norms and rules. The explicit build-up and annotation that involves a user as active part has been avoided by the involvement of multi-domain information resources that are combined to extract a semantic confident relation.

9. Application and Conclusions

By using the actual work context in conjunction with the extracted statistical significant concept relations, matching patterns can be recognized and be used to deliver and apply relevant knowledge to the design process. In the following example (see fig. 7) of the design of a steering wheel, the size of the circles reflects the importance of a certain aspect, as derived from the merged conceptualization of a steering wheel from human and online sources.

The separation of the identified contexts is computed by the occurring frequency of its concepts. The names of the contexts are manually added to mark them as they can be identified through statistical clustering. The focus can now be given to a certain context, depending on where the user is executing the work. If a frequently use of forming concepts is detected, it can be deducted, that the work focus is in a certain design area and knowledge according to related concepts from this context is then presented.

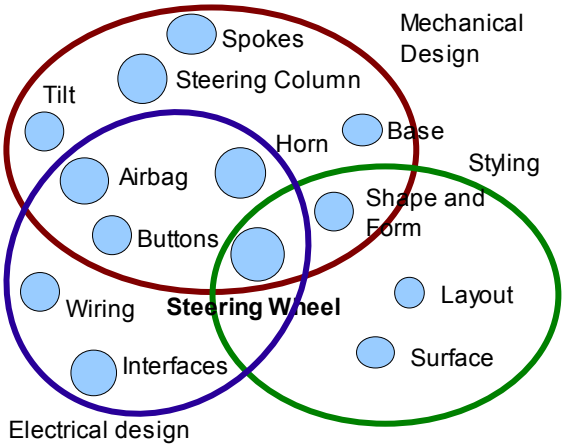


Figure 7. Context related to steering wheel design

Based on a common design context, qualifying situations have to be found in the knowledge base. Depending on the selected semantic method, the system identifies the geometric knowledge that needs to be applied to the user's problem. This information typically consists of a geometric object as discussed in the examples above. It can however be any information aspect that is stored in the knowledge base. After the identification of a geometric solution, it is extracted and transformed into a neutral form, since most certainly the design situation at hand has different parameters and dimensions, so a direct part transfer would not be possible. Out of the neutral form a solution is generated with respect to the actual used parameters. This solution is then proposed to the user who can accept or modify it to his needs.

The design context between two design projects has to match in order to separate relevant from irrelevant solutions. The design context is identified through buildup and use of geometric constructs, additionally to the non geometric information that also delivers characteristics to evaluate the design context. An incorrect design context would yield solutions that are not suited or not optimized for the design situation at hand and would need considerable effort to make them compliant.

If more than one solution has been identified, they have to be evaluated against each other to determine the best fitting expression. This is again based on design context, similarity of design situations and user requirements. The best fitting solution is determined by how strongly it needs to be transformed to the actual design situation and if it fits after the application of the given geometric parameters.

Once a relevant solution is found in another design project in the knowledge base, the solution has to be transformed and applied in a fitting manner into the current design situation. If the solution itself is already reused information or based on a part library, the modifiable parameters are reset and it is transformed into a neutral base form. This base form is then instantiated with the given parameter values.

If the solution is an original part design complex, the modifiable parameters have to be identified before it can be used in the new design project. The transformation takes place on several logical layers. Typical geometric transformations include rotation, translation and scaling, supported by feature transformations, that include the assignment or change of feature types to the involved object. Topological changes regarding the hull or boundary of the object may need to be applied.

A set of anchor points is recorded from the user's actions, that are the driving force in the transformation to decide how to configure the geometry that has to be integrated. The initial application of the solution to the design situation is done by feeding the available parameter information into matched anchor points of the solution object. The solution object is then configured and generates a variant according to its restrictions and parametric dependencies in the knowledge base. If an exact fitting solution cannot be determined, it will be left to the user to make the final adaptations.

For scenarios other than mechanical assembly, there has to be a considerable research effort to capture the domain specifics that are needed for a transforming operation.

10. References

[1] Opletal, S., Roller, D., Rüger, S., "Semantic Core to Acquire and Distribute Design Information", The 2nd Int. Conf. on Advanced Engineering Computing and Applications in Sciences (ADVCOMP'08), IEEE Computer Society Press, 2008

[2] Altmeyer J., Ohnsorge S., et. al., "Reuse of design objects in CAD frameworks", IEEE/ACM Intern. Conf. on Computer-Aided Design, 1994

[3] Baeza-Yates, R., Ribeiro-Neto, B., "Modern Information Retrieval", New York: ACM Press, Addison-Wesley, 1999

[4] Bjørner D., "Domain Engineering" in BCS FACS Seminars, Lecture Notes in Computer Science, BCS FAC Series (eds. Boca, P., Bowen, J.), Springer, 2008

[5] Cardie, C., "Empirical Methods in Information Extraction" in AI Magazine, Vol. 18, 4, 65-68, 1997

[6] Case, K., Graham, I.J., et. al., "CAD Genetic Algorithms for Evolutionary Form and Function Design" in Advances in Manufacturing Technology - XVI , Cheng, K., Webb, D. (Eds), Professional Engineering Publishing Ltd, Proc. of the 18th National Conf. on Manufacturing Research , Leeds Metropolitan University, UK, pp 103-107, 2002

[7] Dalakakis, S., Dieterich, M., Roller, D., Warschat, J., "Multiagentensystem zur Wissenskommunikation in der Produktentstehung-Rapid Product Develoment". Wirtschaftsinformatik 2005, "eEconomy eGovernment eSociety", Ferstel, O.K., Sinz, E.J., Eckert, S., Isselhorst, T. (eds.), Physica-Verlag, 2005

[8] de Moya-Anegón et. al., "Domain analysis and information retrieval through the construction of heliocentric maps based on ISI-JCR category cocitation", in Information Processing and Management 41, Elsevier, 2005

[9] Fischer, M., Staub, S., "Constructability reasoning based on a 4D facility model", in Proc. of Structural Engineering Worldwide. Elsevier Science Ltd., 1998

[10] Flycht-Eriksson A., "A domain knowledge manager for dialogue systems", in Proc. of ECAI 2000, ed. Horn W., Gesellschaft für Informatik, Berlin, 2000

[11] Google Search, Google inc., Mountain View, USA http://www.google.com/advanced_search?hl=en

[12] Kitamura, Y., et. al., "Functional Metadata Schema for Engineering Knowledge Management" in Proc. of WWW 2005, Chiba, Japan, 2005

[13] Macquarie, P. B., "Tacit Knowledge in Organizational Learning and Social Aspects of Technology", IGI Publishing, Hershey, USA, 2008

[14] Leonard, D., Sensiper, S., "The role of tacit knowledge in group innovation", California Management Review, Vol. 40 No. 3, 1998

[15] Pratt, M., Regli, W.C., et.al., "Manufacturing Feature Recognition from Solid Models: A Status Report", in IEEE Transaction on Robotics and Automation, Vol. 16, No. 6, 2000

[16] Mok, C. K., Hua, M., et.al., "A hybrid case-based reasoning CAD system for injection mould design", Int. Journal of Production Research, 1 – 18, 2007

[17] Ogden C.K., Richards I.A., "The Meaning of Meaning: A Study of the Influence of Language upon Thought and of the Science of Symbolism", 8th ed.; New York: Harcourt, Brace &World, 1946

[18] OpenCyc Ontology, OpenCyc Knowledge Base Copyright 2001-2008, Cycorp, Inc., Austin, TX, USA http://www.opencyc.org/

[19] Opletal, S., Dalakakis, S., Roller, D., "Towards Semantic-Based CAD user Interface and Core Components", in "Applications of Digital Techniques in Industrial Design Engineering", Proc. of the 6th Int. Conf. on Computer-Aided Industrial Desingn & Conceptual Desingn, Eds: Pan, Y., Vergeest, J., Lin, Z., Wang, Ch., Sun, S., Hu, Z., Tang, Y., Zhou, L., Int. Academic Publishers, Beijing World Publishing Corporation, Beijing, 2005

[20] Opletal S., Roller D., et.al., "Pro-active environment for assisted model composition", Proc. of Cooperative Design, Visualization, and Engineering (CDVE07), Springer, 2007

[21] Piskorski, J.; Xu F., et.al., "A Domain Adaptive Approach to Automatic Acquisition of Domain Relevant Terms and their Relations with Boot-strapping", in Proc. of the 3rd Int. Conference on Language Resources an Evaluation (LREC), 2002

[22] Ponsen, M., Munoz-Avila, H., et.al., "Automatically Acquiring Domain Knowledge For Adaptive Game AI Using Evolutionary Learning" in Science of Computer Programming, Volume 67, Issue 1, pp. 59-75 Special Issue on Aspects of Game Programming, 2007

[23] Rembold, U., Nnaji, B.O., Storr A., "Computer Integrated Manufacturing and Engineering", Addison-Wesley, UK, 1994

[24] Roller, D., Eck, O., Dalakakis, S., "Knowledge based support of Rapid Product Development" in Journal of Engineering Design, Taylor & Francis Ltd, Vol. 15, No. 4, 2004

[25] Singh V., "The Cim Debacle: Methodologies to Facilitate Software Interoperability", Springer, 1997

[26] St-Cyr O., Yves Lespérance Y., et.al., "An Intelligent Assistant for Computer-Aided Design" Proc. of AAAI 2000 Spring Symposium Series on Smart Graphics, Stanford, USA, 2000

[27] Exemplary Wikipedia page on "sheet metal", Wikimedia Foundation Inc., San Francisco, USA http://en.wikipedia.org/wiki/Sheet_metal

[28] Feiyu, X., Uszkoreit, H., Li H., "A Seed-driven Bottom-up Machine Learning Framework for Extracting Relations of Various Complexity" in Proc. of ACL 2007, Prague, 2007

Semantic Enabled Framework for SLA Monitoring

Kaouthar FAKHFAKH[1-2-3], Saïd TAZI[1-2], Khalil DRIRA[1,2]
[1]CNRS- LAAS
[2]Université de Toulouse; UT1, UPS, INSA, INP, ISAE; LAAS;
7 avenue du colonel Roche, F31077 Toulouse, France
{kchaari, tazi, khalil}@laas.fr

Tarak CHAARI[3] and Mohamed JMAIEL[3]
[3]National Engineering School of Sfax
ENIS-ReDCAD
Informatics and applied Mathematics department,
Route de la Soukra, B.P. W, 3038 Sfax, Tunisia
tarak.chaari@redcad.org
mohamed.jmaiel@enis.rnu.tn

***Abstract*--Defining clear Quality of Service agreements between service providers and consumers is particularly important for the successful deployment of service-oriented architectures. The related challenges include correctly elaborating and monitoring QoS-aware contracts (called SLA: Service Level Agreement) to detect and handle their violations. In this paper, first, we study and compare existing SLA-related models. To address the insufficiencies of these models, we propose a complete, generic and semantically richer ontology-based model of Service Level Agreements. In this model, we use the SWRL language (Semantic Web Rule Language) to express SLA obligations. This language facilitates the SLA monitoring process and the eventual action triggering in case of violations. In a second step, we use our SLA model to automatically generate semantic-enabled QoS obligations monitors. The main algorithms that perform the monitoring process are presented in this article. We implement these algorithms in an automatically generated service-oriented architecture. Finally, we believe that this work is a step ahead to the complete automation of SLA management process.**

Keywords—Service Level Agreements; ontology-based model; SOA; SLA monitoring; QoS contracts

I. INTRODUCTION

Service Level Agreements (SLAs) have become very important in the information technology area of business firms. SLAs are used with increasing frequency in general application integrations, e-commerce, outsourcing and B2B deployments. As firms increased their outsourcing of IT services, SLAs become the primary management tool for governing the relationship among the provider and its consumers. The emergence of software as a service, especially Web service, has also spurred the development of service level agreements. As more business software moves to a Web delivery platform, SLAs became the primary tool that regulates the relationship between providers and consumers when they use software services.

Metrics like processing time, messages per hour, rejected transaction counts and queries per day are common examples of defined service qualities which may be measured either at end-points, or by an intermediary. These measurements are then typically compared by an enforcement process or application to the desired level. An action should be taken according to this comparison. This action can be simply gathering and reporting results, identifying and forwarding SLA violations, or changing service behavior based on current SLA conformance.

Monitoring of SLAs between providers of a service (for example on-line banking, auctioning, ticket reservation, etc.) and consumers is a topic that is gaining in importance for business success over the Internet. SLA monitoring involves the collection of statistical metrics about the performance of a service to evaluate whether the provider is delivering the level of QoS stipulated in a contract signed between the provider and the consumer. In this context, the monitoring and the management of SLAs and their related services are crucially important. Our work focuses on the required models and software tools to monitor the QoS obligations specified in these contracts and to react to the violations or failures in the system. In this paper, we focus on a generic ontology [1] development to assist the preparation of QoS contracts and to monitor the agreements and the specified obligations on these contracts. The choice of ontology is driven by its potential to facilitate the establishment of service level agreements between the different knowledge levels of service providers and consumers. In addition, ontology implementations, using open standards like OWL (Web Ontology Language) [2] and SWRL (Semantic Web Rule Language) [3], provide a common understandable language for machines and humans. They also facilitate the contract obligations expression and the necessary inferring to take the appropriate actions in case of violations.

In Section II of this paper, we start by defining the principles of the service level agreements, their structure, their establishment and their existing implementations. In Section III, we present the main SLA related existing models. In Section IV, we detail our service level agreement's generic model that we called *SLAOnt*. Then, in Section V, we explain how this model is used to monitor its obligation instances. We present also the simplified architecture and the main algorithms that perform the monitoring process. In Section VI, we present the SLA monitoring API (called *SLAOntAPI*) that we have developed to implement the monitoring algorithms. Before concluding, in Section VII, we give a simple instantiation example of our model and we show how we have monitored its obligations using our SLA monitoring prototype.

II. SLA PRINCIPLES

In this section, we present the definition of a Service Level Agreement (SLA), its structure, life-cycle and its main implementing languages that we can find in the literature.

A. Definition: Service Level Agreements (SLA)

Debusmann and al., in [4], define the term SLA as a contract that exists between consumers and their service provider, or between service providers. It records the common understanding about services, priorities, responsibilities, guarantee, and the quality level of the service according to all these parameters. For example, it may specify the levels of availability, serviceability, performance, operation, or other attributes of the service like billing and even penalties in the case of violation of the SLA.

SLA is also described in [5] as a "Contractual service commitment". An SLA is a document that describes the minimum performance criteria a provider promises to meet while delivering a service. It typically also sets out the remedial action and any penalties that will take effect if performance falls below the promised standard. It is an essential component of the legal contract between a service consumer and the provider."

B. SLA structure

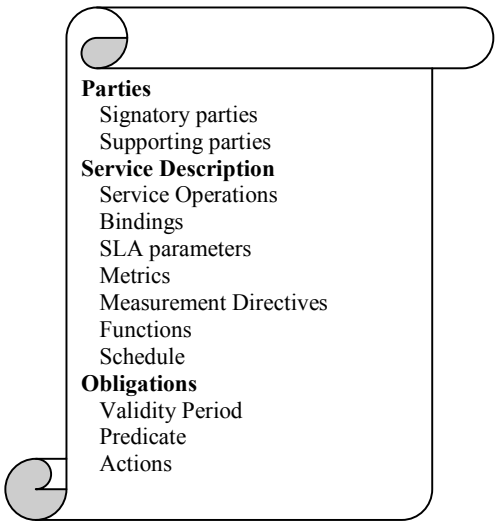


Figure 1. General SLA Structure

SLA is composed of three main sections as presented in figure 1. The first section contains the involved parties in the contract: the signatory parties (the service provider and its consumer) and the third parties that supervise SLA obligations. The second section presents the involved services description. This part contains the service operations, their input and output messages. For each service operation, one or more bindings may be specified. A binding is the transport encoding for the messages to be exchanged. It also contains the SLA parameters representing the QoS variables that will be used in the specification of the contract obligations. These parameters are based on metrics evaluated by measurement directives. Some functions can be used to aggregate multiple metric values. The last element (schedule) of this second part in the contract specifies the duration and the frequency of QoS measurements. The third section presents the contract obligations: their validity period indicating the time intervals for which a given SLA parameter is valid (for examples, business days, regular working hours or maintenance periods), the predicate that represents the conditions that specify these obligations and the actions to be taken when the contract is not respected.

C. SLA life cycle

Although the contracts are intended to formalize mutually accepted agreements by services providers and consumers, their establishment usually remains usually asymmetric and controlled by the providers. It includes several steps as shown in figure 2.

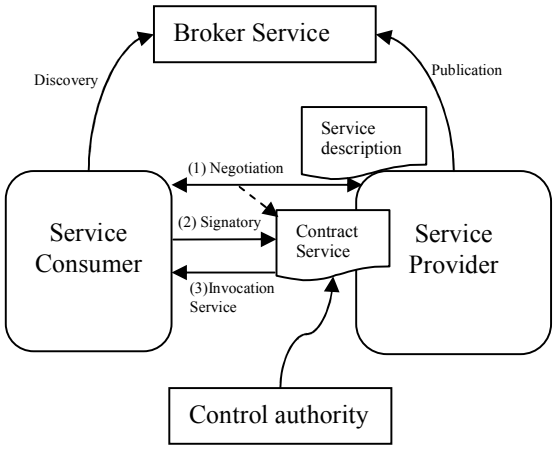


Figure 2. SLA Establishment

The service provider creates a contract model that defines the offered services and their associated constraints. Then, it publishes them at a given broker service. They also integrate the service's financial costs (included in the SLA obligations) as well as penalties in case of contract violation. The service consumer discovers this model from the broker service and selects the desired services and the contract instance. When the provider receives this instance, he checks it before its validation and sends it to the consumer. After a negotiation phase and when the two parties are in agreement, they sign the contract. After that, the consumer can invoke her/his the corresponding service. The specified obligations in the contract are constantly supervised by a controlling authority which is a third party that notifies the signatory parties when the contract is violated.

D. SLA implementations

Several languages were proposed to implement the SLA specifications. We can cite WSOL (Web Service Offerings Language) [6], GXLA [7], WSML (Web Services Management Language) [8], SLAang [9], Ws-Agreement

[10], Ws-Negotiation [11] and WSLA (Web Service Level Agreement) [12]. Among all these languages, the most successful contribution is the WSLA language created by IBM[1]. It is a flexible and extensible language based on XML Schema[2]. However, contract development remains a difficult task to achieve when using this language. In fact, providers and consumers don't have the same degree of knowledge and may not share the same language. In addition, the contract monitoring and its possible violation are difficult to establish due to some insufficiencies in the monitored QoS parameters description especially when they are composed by other elementary parameters. Consequently, we explore the existing SLA related models to find more structured and semantically richer descriptions of SLA obligations to ensure their automatic monitoring and management.

III. SLA-RELATED EXISTING MODELS

In this section, we present an analysis of some SLA related existing models. We notice a great interest in modeling the quality of services which is a principal element in the contract specification. In the following parts of this section, we present the main existing models in QoS specifications.

A. OWL-QoS (Chen Zhou, Likang-Tien Chia, Bu-Sung Lee)

OWL-QoS [13] is a QoS description model. It reuses OWL-S [14], the service description ontology standard. This model is characterized by its formal QoS specification, distribution and consumption. Unfortunately, it presents some insufficiencies: QoS metrics are instantiated without specifying how they will be measured and in what context they can be used. Moreover, the used approach is flawed in that it uses cardinality constraints to express bounds upon QoS properties. As the term cardinality suggests, this is actually a misuse of this OWL construct. A cardinality constraint puts constraints on the number of values a property can take, not on the values themselves. Even if the approach taken was valid, it also carries the limitation that it can only express bounds as positive integers (e.g., there is no simple way to say "availability> 0.999").

B. QoSOnt (G. Dobson, R. Lock, I. Sommerville)

QoSOnt [15] has much in common with other OWL ontologies [16] for web services. It contains links to OWL-S and concentrates on the metrics definition and on QoS requirements matching with metrics. As well as pointing the direction to the correct semantics for matchmaking, QoSOnt also correctly identifies that the value of a metric is only relevant in the correct scope (e.g.. network latency applies to a particular network route) and that metric has a "direction" e.g., the higher, the better. Initial attempts at representing how metrics combine when services are composed have also been made. Unfortunately, despite identifying the correct semantics for matching QoS with its metrics, QoSOnt uses a non standardized XML language losing many of the advantages of OWL [2].

C. SL-Ontology (Steffen Bleul, Thomas Weise, Kurt Geihss)

SL-Ontology [17] is another attempt at QoS modeling. It differentiates between the provider offers and customer demands. It presents the necessary elements of quality aware service discovery and the importance of integrating quality aspects in service integration. A description language needs flexibility for service level packages and service providing parties. It must also handle different terms in specifying QoS-Dimensions. In this scope, SL-Ontology specifies a part of measurement units transformations to address disparities between customers and suppliers languages. This resolution is specified only at the level of units in this model.

D. WS-QoS (Tian, M., Gramm, A., Ritter, H., and Schiller, J.)

WS-QoS [18] is a framework that uses a QoS-based ontology model for the dynamic Web services selection depending on the performance requirements and network bandwidth. This model is characterized by specific metrics that must be known in advance by all the services. It also uses a specific non-OWL XML language for metric description. Consequently, it loses the reasoning and the semantic inferences offered by the OWL language.

E. FIPA QoS (M.B Alberto, G.V Marisol)

FIPA [19] is another ontology-based model of QoS representation. It is complete, but unfortunately it remains too specific to the lower layers of the OSI model. This ontology also lacks an openly available implementation and links to OWL-S ontology. It has also been applied only in FIPA architecture and therefore it is not directly applicable in a web services environment.

F. MOQ (HM. Kim, A. Sengupta and J. Evermann)

MOQ [20] is another attempt of QoS modeling that defines QoS composite requirements but fails to suggest a mean to allow logical requirement combinations, only stating that if all sub-requirements are met then the composite is always satisfied. Unfortunately, the major drawback of MOQ is that it does not in itself seem to present an ontology, but only talks about the semantics of QoS ontologies in general. It doesn't use a vocabulary or taxonomy of QoS terms in its modeling and therefore it fails to address all of the issues that complete ontologies.

G. Synthesis on existing QoS models

We have made a comparative study between these various models. Table 1 presents a comparison of these models according to three criteria. The first criterion "Scope" illustrates the degree of completeness of each model by listing its main concepts. The second criterion "Implementation" shows if concrete examples were developed to validate these models. The third criterion

[1] http://www.research.ibm.com/wsla/

[2] http://www.w3.org/XML/Schema

"Automatic use facilities" illustrates the degree of information structuring into these models to facilitate their interpretation and their automatic use to monitor and manage service level agreements.

TABLE 1. QOS MODELS COMPARISON

	Scope	Implementation	Automatic use facilities
OWL-QoS [13]	Metric, Unit, Measurement functions, QoS Profile, agreements, Actors, Service	Partial implementations	QoS constraints as strings
QoSOnt [15]	Service Profile, QoS Profile, Metric, Unit, Actors, Measurement functions, Service	Partial implementations	Specific partial implementation language
SL-Ontology [17]	Metric, Unit,, QoS, Services	Partial implementations	for SLA establishment but not for monitoring
WS-QoS [18]	Metric, Functions, QoS, agreements, Actors	No OWL implementation	Specific implementation language
FIPA QoS [19]	Quality of Service Description, Rate Value, Probability Value, Transport Protocol Description	No OWL implementation	Specific implementation language
MOQ [20]	Qos Requirement, Traceability, Management	Provides a theoretical basis without implementations	Specific predicate interpretation

On the first criterion, the majority of the existing models have focused on the specification and the measurement of QoS. Few models are interested in establishing and managing the QoS contracts. Consequently, we usually have incomplete specifications to express the obligations of the involved actors in the contracts. On the second criterion, we looked for concrete examples that instantiate the existing models. We encountered various difficulties with the majority of them due to the insufficiency at the level of QoS obligation specifications in the contract. For example, the MOQ model is very abstract and lacks many concrete elements to be really implemented in real world instances. On the third criterion, specific and non standard implementations of some parts of all the described models make their automatic interpretation and monitoring difficult to establish. Ad-hoc solutions have to be developed to use these models.

All the models that we cited have advantages and relative limitations. Indeed, few existing models define the context concept in the quality of service (QoS); however, context is important to manage the contract lifecycle for QoS in an automatic way. In addition, some contributions (as WS-QoS) use specific XML formats for the full or partial implementation of their models. This may reduce the interest of using this ontology. In fact, the ontology offer inference possibilities and semantic interpretations when they are implemented using the OWL language. In addition, some models are either specific to a particular domain such as FIPA-QoS which is specific to the low layers of OSI model) or presenting various insufficiencies (like the lack of specifications of logical constraints in MOQ). Finally, all these models focus on the quality of service modeling without detailing the obligations and agreements between the involved actors. This last observation motivated us to develop an SLA model based on the advantages of the existing contributions.

Our contribution in this domain is to establish an ontology-based service level agreements (SLA) model. We made this choice to (i) facilitate the establishment of contracts between entities (suppliers and consumers) having different knowledge levels (ii) have a model offering rich semantics to be understood by humans and by machines (iii) use the semantic richness of SWRL rules in order to express SLA obligations and to easily infer and directly apply the necessary actions in case of violations and (iv) use their semantic richness to diagnose the causes of these violations.

IV. SLAONT: ONTOLOGY-BASED SLA MODEL PROPOSITION

Our model, that we called SLAont [21], defines an ontology describing various concepts and properties needed in a quality of service contract. Figure 3 presents the generic structure of this model. The root is the SLA concept. It represents the contracts class that can be instantiated from *SLAOnt*. This class is composed of the following concepts: Parties, Obligation and *ServiceDefiniton*. The first concept *Parties* defines the involved parties in the contract: the signatory and the supporting party. The signatory parties are generally the service providers and their consumers. The third parties provide the necessary entities for the quality of service measurement evaluation and monitoring. The second concept Obligation defines the quality of service obligations that have to be respected by the parties. These obligations are defined by service level objectives. Each objective is composed of predicates describing the QoS clauses that may cause the contract violation. The third concept *ServiceDefinition* describes the provided services that are concerned by these obligations. Our model uses the OWL-S [14] ontology to describe these services. This ontology is composed of three main parts: the service profile for advertising and discovering services; the process model, that gives a detailed description of the service operation; and

the grounding that provides details on how to interoperate with a service, via messages.

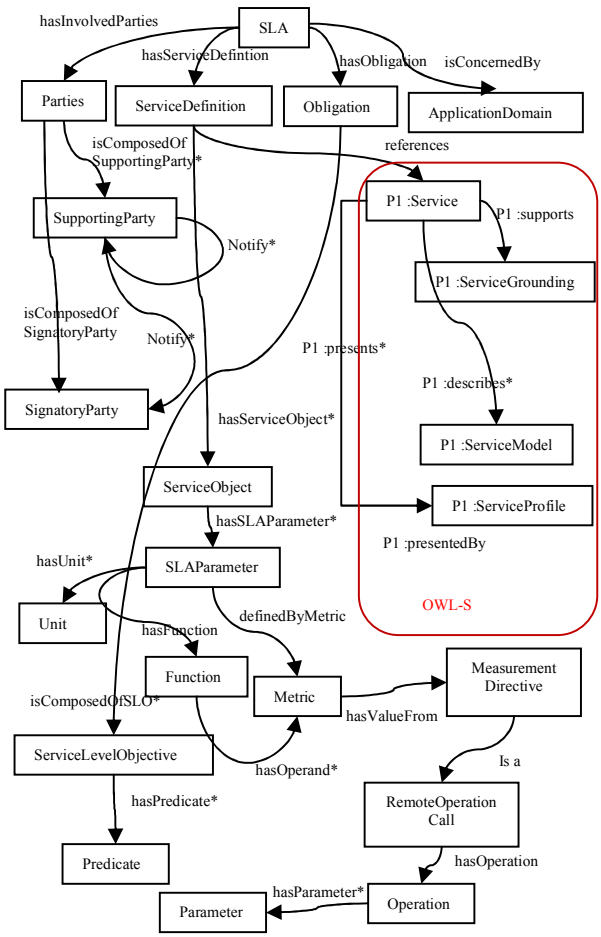


Figure 3. *SLAOnt*: general structure of a QoS contract

The contract is defined in a specific application domain. Thus, the concept *ApplicationDomain* is defined to describe the contract context. This concept is a reference to an external ontology describing the business domain of the involved service in the SLA. The necessary variables for defining the measured quality of service are modeled by the *SLAParameter* concept. *SLAParameter* is associated with one or more metrics (an aggregation of metrics), that define the quality of service parameters to be measured in the contract. These metrics, modeled by the metric concept, can be aggregated by a mathematical function or an algorithm defined in the Function concept. For example, the function can be an average or a computed percentage from a set of measurements.

The *SLAParameter* concept represents a variable that has a measurement unit (Unit) that can be seconds, minutes, percentages, etc. For each SLA parameter, the SLA designer has to specify the corresponding aggregation function to compute its values. These functions are described in the SLA instance. They can be composite and they are described as an abstract service providing an SLA parameter value and consuming metrics values or constants. Each function is defined by an implementing class and a list of operands corresponding to measurements of QoS metrics. These measurements are obtained using measurement directives *MeasurementDirective* corresponding to remote operation calls to retrieve the metrics values. We have modeled these calls by the *RemoteOperationCall* concept. It describes the operation to be called with its invocation protocol. Each operation is associated to a handler that represents the remote class that hosts the operation. In addition, we describe input parameters of the operation with the Parameter concept. This encapsulates the values to be passed to the remote operation for its invocation. The Predicate concept defines the QoS obligations that must be respected in the contract. Each predicate is expressed by an SWRL rule. Each rule is defined by a head part (swrl:head) and a body part (swrl:body). In the head part, we specify the actions that have to be taken when a violation is detected. The body part specifies the conditions that may cause a contract violation. Listing 1 gives a concrete example of an SLA predicate in SLAOnt. This rule sends a disseminate violation message to the signatory parties if the service response time is greater or equal to 100 milliseconds.

Listing 1. Response time less than100 ms evaluation rule

```
hasEvaluation (response_time, ?x) ^ swrlb: greater ThanOrEqual(?x, 100.0) →
slaont:disseminateViolation (response_time, ?x)
```

SLAOnt is a generic and rich model in terms of semantics. In fact, *SLAParameters* can be semantically composed of metrics according to a user defined aggregation functions. For example, an SLA parameter can describe the average response time of a service. In this case, the attached metric is response time and its aggregation function is the average function. Moreover, measurement directives are generic enough to evaluate any metric defined in this model. The model can also express dependencies between metrics, *SLAParameters* and QoS obligations. These dependencies can be statistical (aggregation to calculate average for example), logical (comparison with many thresholds) or semantic (parameters deduced by inference from other parameters). Finally, the semantic relationships between metrics, *SLAParameters* and QoS obligations offer an easy and reliable means to (1) evaluate them and (2) to produce inferences and reasoning in order to detect contract violations and even QoS degradations. In the next section, we present how we used SLAOnt to monitor QoS contracts.

V. MONITORING SLAONT INSTANCES

We used *SLAOnt* model to monitor the obligations defined in the SLAs. In this section, we present the simplified architecture and the main algorithms that perform the monitoring process. Figure 4 shows an overview of this

monitoring process in our approach. It illustrates the main entities of our monitoring API. This API contains a monitoring main module that can deploy *SLAOnt* instances. For each metric, SLA parameter and obligation defined in an *SLAOnt* instance, the main module respectively generates a metric measurement service, an SLA parameter measurement service and obligation measurement service. In the remaining parts of this section, we detail the main functions and algorithms of these services.

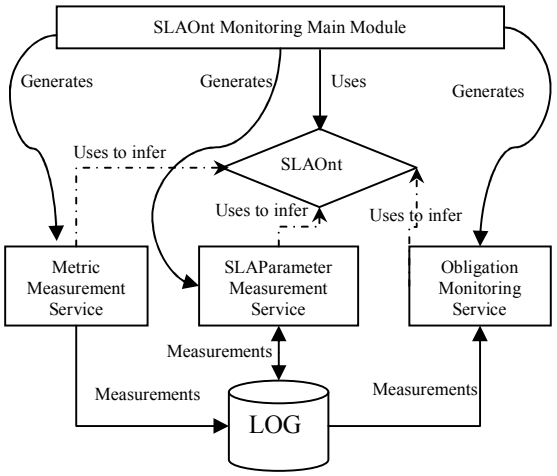


Figure 4. Simplified architecture of *SLAOnt* obligations monitoring

A. Metric measurement services

Metric measurement services provide metric values according to a frequency specified in the SLA. A metric measurement service is automatically instantiated by the *SLAOnt* monitoring main module for each metric defined in the SLA. This service invokes the measurement directive of the associated metric. Listing 2 presents the main functions of this service. First, it collects the metric name, the measurement directive and the measurement frequency of the associated metric. Then, it loops according to this frequency to invoke the measurement directive of the metric and to store the obtained value in a log file. These values will be used by the SLA parameter measurement services.

Listing 2. Metric measurement algorithm

```
MetricMeasurement(Metric metric)
{
 getMetricDetailsRule:="hasName("+ metric + ", ?metricName) ^
   hasValueFrom("+metric+",?measurementDirective)^
   hasMeasurementFrequency("+ metric +",?frequency) →
   query:select(?metricName, ? measurementDirective, ? frequency)";
(metricName, measurementDirective, frequency):= runRule(getMetricDetailsRule);
 Loop on frequency
 {
  measurement := invokeMeasurementDirective(measurementDirective);
  Log.store(metricName,measurement);
 }
}
```

B. SLA parameter measurement services

The SLA parameter measurement services apply aggregation functions on the metrics values to compute the QoS variables defined by the SLA parameter. An SLA parameter measurement service is automatically instantiated by the *SLAOnt* monitoring main module for each SLA parameter defined in the contract. For each SLA parameter, we have to specify the corresponding aggregation function to compute its values. The implementing class of each function will be called by the SLA parameter measurement algorithm after getting the necessary metric values to compute the function. This algorithm collects the different metrics associated with the SLA parameter. It also gets the computation frequency and the aggregation function of the SLA parameter. Then, it loops according to this frequency in order (i) to collect the last measured values of the metrics associated to the SLA parameter, (ii) to compute its aggregation functions and (iii) to add its value in the Log storage. This value will be used by the obligation monitoring services. Listing 3 presents the main functions of this service.

Listing 3. SLA parameter measurement algorithm

```
SLAParameterMeasurement(SLAParameter parameter)
{
 getMetricsRule := "hasFunction("+parameter+",?function) ^
   hasOperand(?function, ?metric) →query:select(?metric);
 metrics := runRule(getMetricsRule);
 getSLAParameterDetailsRule:= "hasFunction("+parameter+",?function) ^
   hasAggregationFrequency("+ parameter + ", ? aggregationFrequency)
   →query:select(?function, ?aggregationFrequency)";
 (function, aggregationFrequency):=runRule(getSLAParameterDetailsRule);
 Every aggregationFrequency do
 {
  Measurements:= ∅;
  For each metric in metrics do
  {
   Measurements.put(metric, Log.getLastValues(metric));
  }
  functionClass := loadFunctionClass(function) ;
  slaParameterValue := functionClass.call(Measurements);
  SLAOnt.setLastParameterValue(parameter, slaParameterValue);
  Log.store(parameter, slaParameterValue);
 }
}
```

C. Obligation monitoring services

The obligation monitoring services check the validity of each obligation defined in the contract. They are automatically instantiated by the *SLAOnt* monitoring main module for each Obligation defined in the SLA. It uses the computed values in the SLA parameter to check if they satisfy the specified conditions defined in the obligation. These conditions are defined as SWRL rules.

Listing 4. Predicate evaluation rule

```
hasEvaluation(average_response_time, ?x) ^  swrlb: greaterThanOrEqual(?x,
    100.0) → slaont:disseminate Violation(average_ response_time, ?x)
```

The evaluation of these conditions is simply an inference of these SWRL rules. The actions to be taken in case of

violations can be directly applied in these rules like *disseminateViolation* in Listing 4.

Listing 5. Obligation monitoring algorithm

```
checkObligation(Obligation obligation)
{
getPredicatesRule:="isComposedOfSLO("+obligation+",?slo) ^ hasPredicate(?slo,
    ?predicate) → query:select(?predicate)";
Predicates: = runRule(getPredicatesRule) ;
For each p in Predicates
 { getPredicateRule := "hasRule("+p+", ?rule) ^
   hasVerificationPeriodicity("+p+",?periodicity) →
   query:select(?rule,?periodicity)";
   (rule,periodicity) := runRule(getPredicateRule)
  Loop on periodicity
   {
    runRule(rule);
   }
 }
}
```

Listing 5 presents the main functions of this service. It starts by collecting the different predicates defined in the associated obligation. Then, every predicate evaluation periodicity (every sixty minutes for example), the inference engine computes the attached SWRL rule and executes the specified action to be taken in case of violation. For example, the SWRL of Listing 1, if the response time is greater than 100 ms, the action *disseminateViolation* is triggered. This action continuously reports all the detected violations and their causes to the involved parties in the SLA.

When the designer creates an *SLAOnt* instance, she/he can specify an execution order for the SWRL rules representing the SLA predicates. This order is ensured by a numbering sequence in the name of the rules that should be conflict free in order to produce relevant results. This conflict verification should be performed before the monitoring phase. The verification process is out of the scope of this work. Actually, we are working on a negotiation approach that generates SLAOnt instances with conflict free obligations.

In the next section, we present how we implemented these algorithms to develop a complete and a reusable monitoring API for *SLAOnt* instances.

VI. SLAONT MONITORING API

In this section, we present the *SLAOnt* monitoring API (named *SLAOntAPI*) that we have developed to implement the algorithms presented in the previous section. Figure 5 shows the technical architecture of the monitoring process. In the lowest layer, ontologies used in the API are represented by their OWL files. Above this layer, the Xerces XML API[3] is used to read data from the owl file. Protégé OWL API[4] is used to handle owl data in the ontologies. Then, the SWRL Jess API[5] is used to make inferences and reasoning on the ontologies instances. In the upper layer, we have developed a JAVA Monitoring API (*SLAOntAPI*).

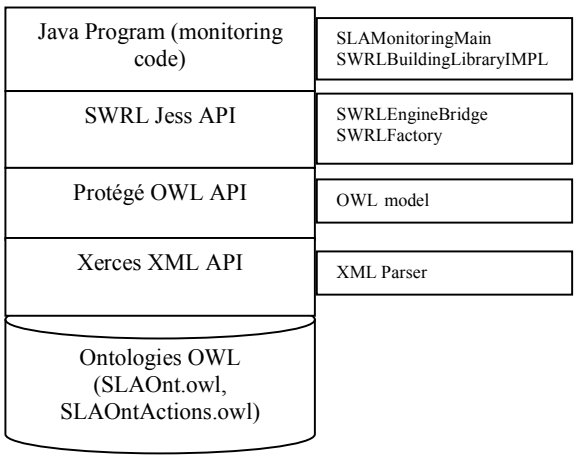


Figure 5. Technical architecture of the monitoring process

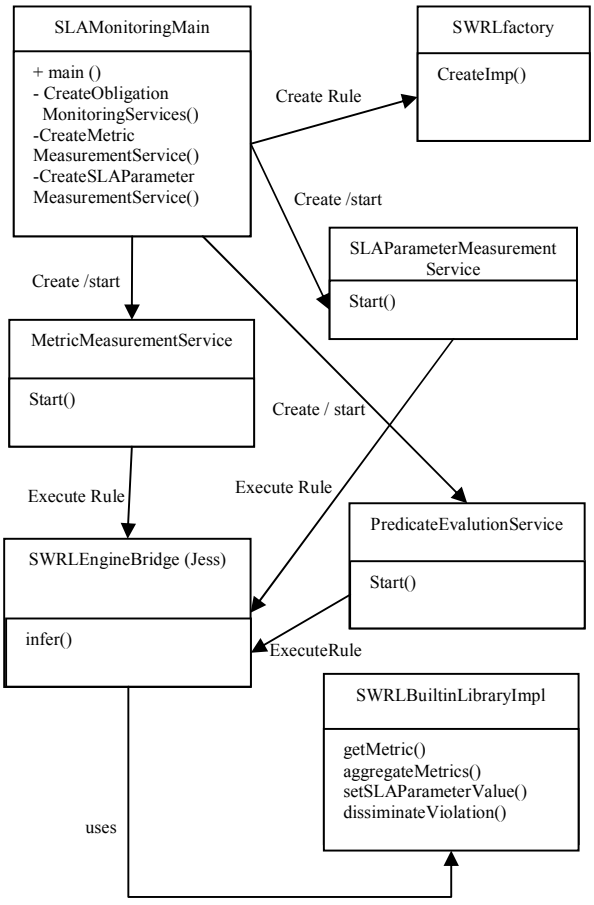


Figure 6. The *SLAOntAPI* class diagram

[3] http://xerces.apache.org/xerces-j/apiDocs/index.html

[4] http://protege.stanford.edu/plugins/owl/api/

[5] http://protege.cim3.net/cgi-bin/wiki.pl?SWRLJessTab

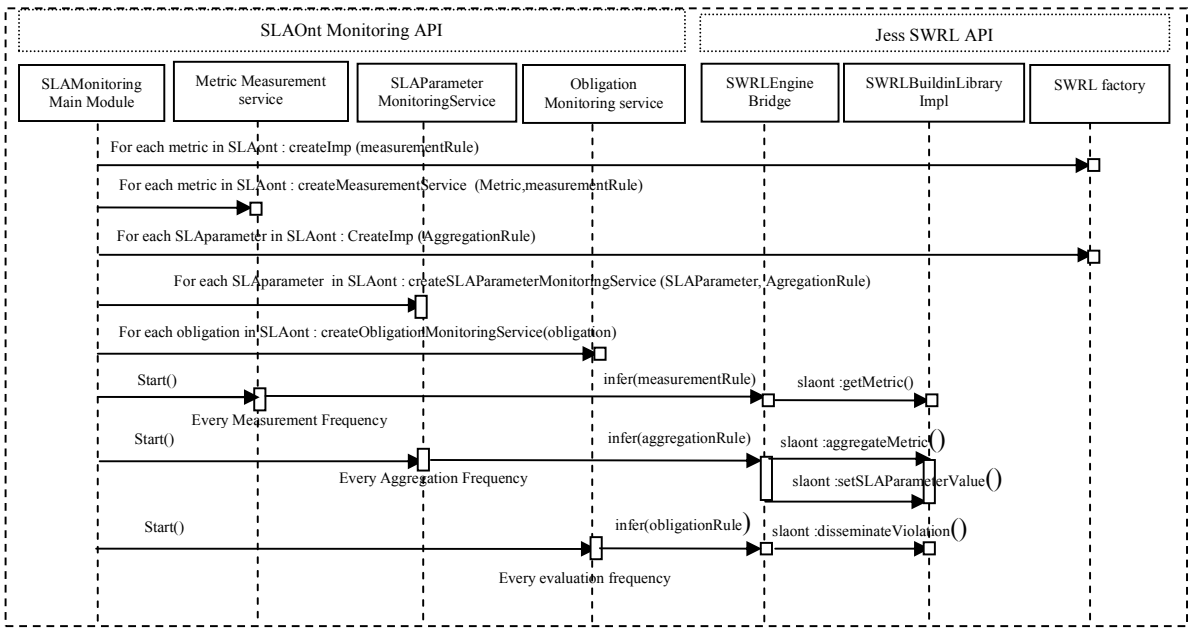


Figure 7. The *SLAOntAPI* Sequence Diagram

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:swrl="http://www.w3.org/2003/11/swrl#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns="http://www.laas.fr/~kchaari/slaOntActions.owl#"
  xml:base="http://www.laas.fr/~kchaari/slaOntActions.owl">
  <owl:Ontology rdf:about=""/>
  <rdf:Property rdf:ID="inArgs"/>
  <owl:DatatypeProperty rdf:ID="minArgs"/>
  <owl:DatatypeProperty rdf:ID="maxArgs"/>
  <owl:DatatypeProperty rdf:ID="args"/>
  <swrl:Builtin rdf:ID="disseminateViolation">
    <minArgs rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
    >1</minArgs>
  </swrl:Builtin>
  <swrl:Builtin rdf:ID="getMetric">
    <args rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</args>
  </swrl:Builtin>
<swrl:Builtin rdf:ID="aggregateMetric">
    <minArgs rdf:datatype="http://www.w3.org/2001/XMLSchema#int">2</minArgs>
  </swrl:Builtin>
```

Figure 8. OWL file containing the actions to be triggered in SLA predicates

Figure 6 shows the class diagram of the *SLAOnt* API. The most important class in this API is the *SWRLBuiltInLibraryImpl* class. It contains the functions that will be directly invoked by the SWRL rules. This class offers the *getMetric* function which invokes the measurement directive of the specified metric and sends its value to the Log store. It also has an *aggregateMetrics* method that calls the aggregation function to compute SLA parameter values. Finally, it offers a *disseminateViolation* method that sends SLA violation messages to the signatory parties. These methods are defined as SWRL Built-ins which are predicates that can take on or more arguments.

Built-ins are analogous to functions in production rule systems. A number of core built-ins are defined in the SWRL specification. This core set includes basic mathematical operators and built-ins for string and date manipulations. These built-ins can be used directly in SWRL rules. User defined built-ins has to be declared in an external ontology. We have declared *getMetric*, *aggregateMetrics* and *disseminateViolation* built-ins in the ontology represented by an Owl file (slaOntActions.owl) as shown in figure 8.

Figure 7 shows the sequence diagram of our approach. For each metric in SLAOnt, the main program creates a measurement rule and a measurement service. For each *SLAParameter*, The main program creates a metric aggregation rule and a service to perform the aggregation. For each obligation in *SLAOnt*, the main program generates an obligation evaluation service.

VII. CASE STUDY: THE FLIGHT SLA EXAMPLE

To validate the service level agreements model and the developed monitoring API, we created an instance of *SLAOnt* model using the "protégé" tool. This instance consists in a simple agreement example between a provider of a flight booking service and its consumers. This service must provide an average response time less than 100 milliseconds for a certain class of clients. Figure 9 illustrates the *FlightSLA* instance in this example.

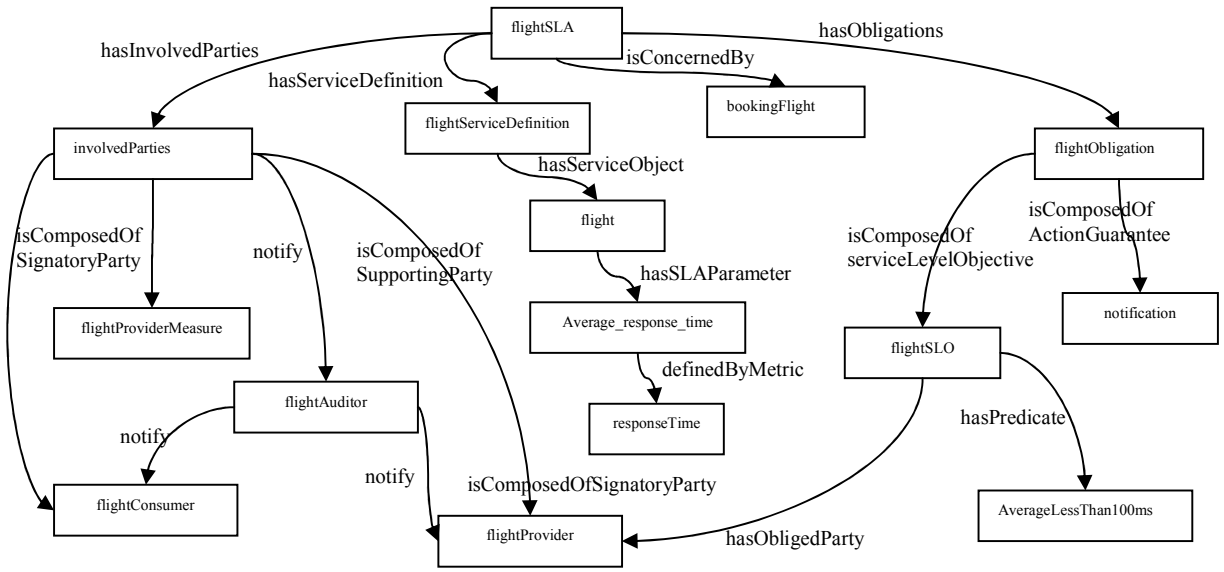


Figure 9. FlightSLA: An SLAOnt instance example

The service provider of this contract is named *FlightProvider* and his consumer is *FlightConsumer*. In this example, the third parties involved in this contract are: *FlightProviderMeasurement* and *FlightAuditor*. The *FlightProviderMeasurement* service must provide the response time measurements of the consumer requests. The *FlightAuditor* service must notify any violations of the contract to the signatory parties. The service provider must respect the objective defined by the *FilghtSLO* instance in the contract. This instance specifies the predicates that have to be satisfied in the contract. These predicates are defined in SWRL rules to facilitate the monitoring processes of the contract. The defined obligation in the *FlightSLA* example is shown in Listing 6.

Listing 6. Predicate evaluation rule

```
hasEvaluation(average_response_time, ?x) ^ swrlb: greaterThanOrEqual(?x,
100.0) → slaont:disseminate Violation(AverageLessThan100ms, "false", average_
response_time, ?x)
```

This rule verifies that the average response time of the monitored service is greater or equal to 100 milliseconds (body part of the rule). In this case, a violation message is disseminated to the signatory parties in the contract. These messages contain the parameter values that caused the violation. To perform the automatic monitoring process on this example, we loaded its owl file[6] in the monitoring API main module (figure 10).

Figure 11 shows the monitoring process of this example. To use the *SLAOntAPI*[7] with other SLA instances, the SLA designer should import the *SLAOnt* ontology[8] and creates the necessary instances of its main concepts. The actions to be taken in case of violations should be declared in an external ontology named *SLAOntActions.owl*. These actions should be implemented as SWRL Built-ins to work with our code. These built-ins are standard java code that can be easily personalized to manage the actions that should be taken in case of violations. Finally, the designer should save the created instance in an owl file and load it in our monitoring main module as shown in figure 10.

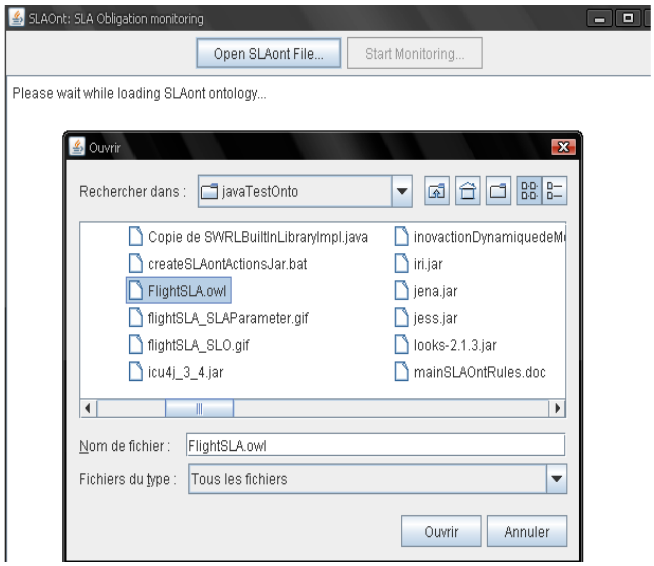


Figure 10. SLAOnt instances loading interface

[6] http://www.laas.fr/~kchaari/slaOnt/FlightSLA.owl

[7] http://www.laas.fr/~kchaari/slaOnt/SLAmonitoring.zip

[8] http://www.laas.fr/~kchaari/slaOnt/SLAont.owl

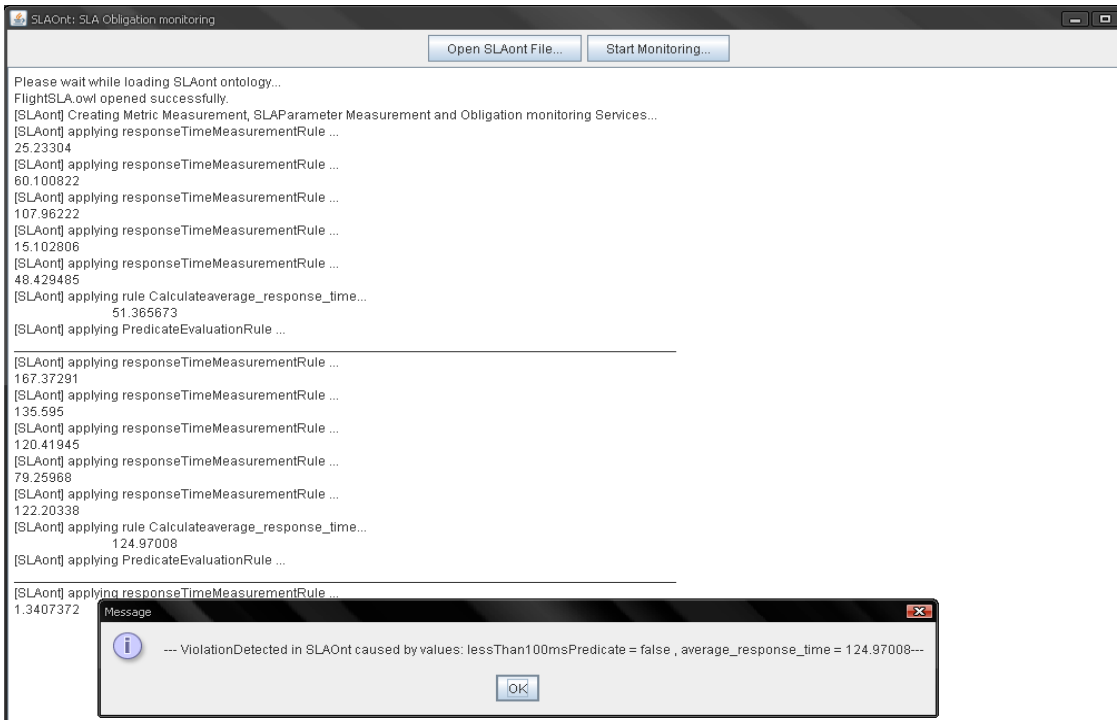


Figure 11. FlightSLA monitoring process

VIII. CONCLUSION AND FUTURE WORK

The service oriented software engineering market requires QoS specifications by the services suppliers. Much effort has been invested in modeling QoS parameters to allow an automatic (or semi-automatic) selection of the services offering the best quality. We have explored several existing models in this domain. We have noticed the lack of a comprehensive and a generic model for the service level agreements specification and for their monitoring to detect possible violations. Therefore, we created an ontology that models these agreements (SLA) to facilitate the QoS contracts establishment between consumers and suppliers on one hand and to automate their management and monitoring on the other hand. In this paper, we presented the structure of our ontology-based model offering rich semantics to be understood by humans and by machines. In this model, we used the semantic richness of SWRL rules in order to express SLA obligations and to easily infer and apply the necessary actions in case of violations. Our second contribution in this work is the development of an API that guarantees the automatic monitoring of our SLA model instances. This API is based on an automatic generation of a service oriented architecture that gathers the measurements of the QoS defined in the SLAs. For clarity reasons, we presented a simple SLA example to illustrate the main principles of our SLA model and our monitoring API. We have tested this API on more complex examples concerning video streaming provider who offers two services: the first one to visualize film online and the second one for downloading films. In this example, a download time SLA parameter is monitored according to the video size and the client's throughput. Our API is scalable enough to handle a large number of metrics SLA parameter and obligations. In fact, their associated measurement services are instantiated dynamically in separate threads and can be distributed on many machines. We plan to use the monitored measurements to analyze and detect system degradations and to prevent SLA violations. Actually, we are working on a semantic-enabled negotiation framework to help the providers and their customers in establishing *SLAOnt* contracts. In a long term future work, we intend to propose corrective actions in case of QoS degradation. This issue will be very useful to evolve from the existing simple message notifications to corrective actions assistance.

ACKNOWLEDGEMENTS

This work has been partially funded by the ITEA2's UseNet (Ubiquitous M2M Service Networks) European project[9].

REFERENCES

[1] R.Studer, R.Benjamins, D.Fensel. Knowledge engineering: principles and methods. IEEE Transactions on Data and Knowledge Engineering, 25 (1-2) pp.161-197, March 1998.

[9] https://usenet.erve.vtt.fi/

[2] Mc Guiness D. & ZSPLITZvan Harmelen F. (2004). OWL Web Ontology Language Overview, W3C Recommendation http://www.w3.org/TR/owl-features/

[3] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. SWRL: A semantic web rule language combining OWL and RuleML. W3C Member Submission, 21 May 2004.

[4] M. Debusmann, R. Kroger, and K. Geihs. "Unifying Service Level Management Using an MDA-based Approach". IEEE Network Operations and Management Symposium, pp.801-814, 2004.

[5] Procullux Media Ltd. http://looselycoupled.com/glossary/SLA. Last visited 15/01/09

[6] Tosic, V., Pagurek, B., Patel, K., Esfandiari, B., and Ma, W. 2005. Management applications of the web service offerings language . Inf. Syst. 30, 7 (Nov. 2005), 564-586.

[7] Badis Tebbani, Issam Aib. "GXLA a Language for the Specification of Service Level Agreements". Autonomic Networking, 201-214, 2006.

[8] A. Sahai, V. Machiraju, M. Sayal, A. van Moorsel, and F.Casati. "Automated SLA Monitoring for Web Services". IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, pp.28-41, 2002.

[9] J. Skene, D. Lamanna, and W. Emmerich. "SLAng: A Language for Service Level Agreements". Workshop on Future Trends in Distributed Computing Systems. IEEE Computer Society, 2002.

[10] A. Andrieux, A. Dan K. Czajkowski, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne,J. Rofrano, S. Tuecke, and M. Xu. "Web Services Agreement Specification (WSAgreement)". Specification draft, Global Grid Forum (GGF), September Version 09/2005.

[11] P. C. K. Hung, H. Li, and J-J Jeng. "WS-Negotiation: An Overview of Research Issues". Hawaii International Conference on System Sciences (HICSS), 2004.

[12] A. Keller and H. Ludwig. "The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services". Journal of Network and Systems Management, 11(1), 2003.

[13] Chia, Bu-Sung Lee. "QoS Measurement Issues with DAML-QoS". IEEE InterChen Zhou, Likang-Tien national Conference on e-Business Engineering (ICEBE'05) pp. 395 403.

[14] OWL-S. An OWL-based Web service ontology. <http://www.w3.org/Submission/2004/07/> November 2004.

[15] G. Dobson, R. Lock, I. Sommerville. "QoSOnt: a QoS Ontology for Service-Centric System", EUROMICRO Conference on Software Engineering and Advanced Applications, Porto, Portugal, Aug. 2005.

[16] C. Zhou, L. Chia, and B. Lee, "DAML-QoS Ontology for Web Services", Proceeding of the International Conference on Web Services 2004 (ICWS04), San Diego, California, USA, July 2004.

[17] Steffen Bleul, Thomas Weise, Kurt Geihss. "An Ontology for Quality-Aware Service Discovery". Special Edition Editorial: Engineering Design and Composition of Service-Oriented Applications, Computer Systems Science & Engineering, Volume 5, Number 21 – 2006.

[18] Tian, M., Gramm, A., Ritter,H., and Schiller,J. "Efficient Selection and Monitoring of QoSaware Web services with the WSQoS Framework". IEEE/WIC/ACM International Conference on Web Intelligence (WI'04), Beijing, China, 2004.

[19] Foundation for Intelligent Physical Agents "FIPA Quality of Service Ontology Specification", Geneva, Switzerland, Nov. 2002.

[20] HM. Kim, A. Sengupta and J. Evermann. "MOQ: Web Services Ontologies for QOS and General Quality Evaluations", European Conference on Information Systems, Regensburg, Germany. May 2005.

[21] K. Fakhfakh, T. Chaari, S. Tazi, K. Drira, and M. Jmaiel. 2008. A Comprehensive Ontology-Based Approach for SLA Obligations Monitoring. In Proceedings of the 2008 the Second international Conference on Advanced Engineering Computing and Applications in Sciences - Volume 00 (September 29 - October 04, 2008). ADVCOMP 2008. IARIA. Published by the IEEE Computer Society Press, pp. 217-222.

Complex software systems : Formalization and Applications *

Marc Aiguier, Pascale Le Gall and Mbarka Mabrouki
École Centrale Paris
Laboratoire de Mathématiques Appliqués aux Systèmes (MAS)
Grande Voie des Vignes - F-92295 Châtenay-Malabry

Programme d'Épigénomique
523, Place des Terrasses de l'Agora - F-91025 Evry
{marc.aiguier,pascale.legall}@ecp.fr, mabrouki@epigenomique.genopole.fr

Abstract

A mathematical denotation is proposed for the notion of complex software systems whose behavior is specified by rigorous formalisms. Complex systems are described in a recursive way as an interconnection of subsystems by means of architectural connectors. In order to consider the largest family of specification formalisms and architectural connectors, this denotation is essentially formalism, specification and connector independent. For this, we build our denotation on Goguen's institution theory. In this abstract framework, we characterize complexity by the notion of property emergence.

This work is a revised and extended version of Aiguier, Le Gall and Mabrouki (3rd International Conference on Software Engineering Advanced (ICSEA), IEEE Computer Society Press, 2008).

Keywords-*abstract specification language; abstract architectural connector; emergent property; institution; category theory; transition systems; modal first-order logic.*

1 Introduction

A powerful approach to develop large software systems is to describe them in a recursive way as an interconnection of sub-systems. This has then made emerge the notion of architectural connector as a powerful tool to describe systems in terms of components and their interactions [6, 7, 16, 25]. Academic and industrial groups have defined and developed computer languages dedicated to the description of software architectures provided with architectural connectors, called *Architectural Description Language (ADL)*, such as ACME/ADML [17], Wright [5] or Community [15, 16]. The interest of describing software systems as interconnected subsystems is that this promotes the reuse of components either directly taken in a library or adapted by slight modifications made on existing ones. The well-known difficulty with such systems is to infer the global behavior of the system from the ones of subsystems. Indeed, modern software systems are often open on the outside, that is they interact with the environment, composed of interacting subsystems (e.g. active objects which interact together concurrently [3, 27]) or defined by questioning requirements of subsystems (e.g. feature-oriented systems where each feature can modify the expected properties of pre-existing features [18, 4, 26]). Thus, such global systems may exhibit behaviors, that cannot be anticipated just from a complete knowledge of subsystems. Hence, what makes such software systems *complex* is they cannot be reduced to simple rules of property inference from subsystems towards to the global system.

Following some works issued from other scientific disciplinaries such as biology, physics, economy or sociology [10, 13], let us make more precise what we mean by complex systems. A complex system is characterized by a holistic behavior, i.e. global: we do not consider that its behavior results from the combination of isolated behaviors of some of its components, but instead has to be considered as a whole. This is expressed by the appearance (emergence) of global properties which is very difficult, see impossible, to anticipate just from a complete knowledge of component behaviors. This notion of emergence seems to be the simplest way to define complexity. Succinctly, this could be expressed as follows: suppose a system XY composed of two subsystems X and Y. Let us also suppose we have a mathematical function F which gives all potential pieces of information on XY, X and Y, and an operation '+' to combine potential pieces of information of subsystems. If we have that $F(XY) = F(X) + F(Y)$ then this means that

*This work is performed within the European project GENNETEC *(GENetic NeTworks: Emergence and Complexity)* STREP 34952.

the system XY integrates in a consistent manner the subsystems X and Y without either removing or adding pieces of information. Therefore, we can say that the system XY is not *complex* (i.e. the system XY is said to be *modular*). On the contrary, if there exists some $a \in F(X) + F(Y)$ such that $a \notin F(XY)$ or there exists some $a \in F(XY)$ such that $a \notin F(X) + F(Y)$, then there is reconsideration of some potential pieces of information of X or Y in the first case, and appearance of true emergence in the second case. The system XY is then said *complex*.

In this paper, we will study the notion of complex software systems by using formal specifications, that is we will suppose that every part of systems have been specified in a given formalism from which we can infer properties. The system XY will be built from subsystems X and Y by means of an architectural connector c expliciting how the two subsystems are linked together to form the global system $c(X, Y) = XY$, the connector c being implicit in the notation XY. Finally, the function F will give for a specification its whole set of satisfied properties, the so-called *semantic consequences* of specifications usually noted $X^\bullet$, and $F(X) + F(Y) = (X^\bullet \cup Y^\bullet)^\bullet$. Roughly speaking, this last notation consists in saturating the property derivation mechanism, and then represents the fact that $F(X) + F(Y)$ denote the set of all properties which can be derived from the set of properties $X^\bullet$, resp. $Y^\bullet$, associated to X, resp. Y. The notion of complexity being based on the emergence of properties, a general framework dedicated to complex software systems can be defined independently of formalisms, specifications and architectural connectors. Hence, we investigate an abstract form of complexity, by following the paradigm "logical-system independency". The interest here is simple. We can observe, whatever the formalism used to specify softwares, that the same set of notions underlies complexity. These notions are : architectural connector and emergent property. To formalize abstractly these elements, our approach will be based on previous works:

- we will use the general framework of institutions [20] which is recognized as well-adapted to generalize formalisms. The theory of institutions abstracts the semantical part of logical systems according to the needs of software specifications in which changes of signatures are taken into account. The abstraction of the different parts of logical systems is obtained by using some notions of the category theory such as the category of signatures and the two functors to denote respectively the set of sentences and the category of models over a signature (see Section 2 for the complete definition of institutions and some related notions);

- specifications will be defined following the generic approach of specification logics [14]. The interest of specification logics is they unify in the same framework heterogeneous forms of specifications by considering them as simple objects of a category $SPEC$, while handled specifications over institutions are mostly axiomatic (i.e. of the form (Σ, Ax) where Σ is a signature and Ax is a (finite) set of formulas (axioms) over Σ). However, because we are interested by emergent properties, we will adapt/modify specification logics by defining them over institutions in order to focus on specification properties;

- abstract connectors will be defined by using notions of the category theory. The use of category theory has already been applied strikingly to model the architecture of software systems by Goguen [19] and Fiadeiro & al. [15]. It has also been applied to model complex natural systems such as biological, physical and social systems (e.g. Ehresman and Vanbremeersch's works [13]).
  Fiadeiro & al. [16] have proposed an abstract formal denotation of a class of architectural connectors in the style of Allen and Garlan [6], that is defined by a set of roles and a glue specification. Here, we will go beyond by not supposing any structure in the architectural connectors.

Over our abstract notions of specification and architectural connector, we will define the notion of emergent properties according to the two following classes:

1. the ones we will call *true emergent properties* that are properties which cannot be inferred from subsystem properties,

2. and the ones we will call *non conformity properties* that are subsystem properties which are not satisfied by the global system anymore.

In practice, properties of the first form, i.e. true emergent properties, combine knowledge inherited from subsystems. Thus, they are defined in a richer language than the ones associated to each subsystem, and the presence of such emergent properties is quite natural. Conversely, properties of the second form, i.e. non conformity properties, are often the consequences of bad interactions between subsystems. They characterize properties that are satisfied (resp. not satisfied) by a subsystem considered in isolation, but are not satisfied (resp. satisfied) by the global system incorporating the subsystem in question.

A software system will be then said *complex* when emergent properties can be inferred from it. The complexity of systems just means that we do not benefit from the complete knowledge of subsystems we have, to analyze the behavior of the large system. Hence, the recursive approach used to describe the system cannot be used to analyze its behavior. Complex systems can then be opposed to modular systems

which by definition strictly preserve local properties at the global level (see [24] for a state-of-the-art on the modular approach).

The formalizations of system complexity and emergent properties are interesting if they are done in such way to support the characterization of general properties to guarantee when a system is or is not complex. To answer this point, we will give some conditions under which a system is modular. We will then establish two results: in the first one we will give a sufficient and necessary condition to ensure the absence of true emergent properties. In the second result, we will give sufficient conditions based on the categorical notion of adjunctness to ensure the absence of non-conformity properties.

As a result of our generalization defined in this paper, all the notions, results, and techniques established and defined in our abstract framework are *de facto* adaptable to any institution.

The paper is structured as follows: Section 2 reviews some concepts, notations and terminology about institutions. Section 3 defines an abstract notion of specifications over institutions. In Section 4, abstract architectural connectors are defined and classified as complex and modular. The notations of the category theory used in this paper are the standard ones and can be found in [15]. Although all the notions and results given in this manuscript are exemplified by many examples all along the paper, Section 5 illustrates more particularly the abstract framework developed in this paper to reactive component-based systems described by transition systems and combined together through the synchronous product operation.

Note : This manuscript extends the paper published in the proceedings of [1] with expanded definitions, new results and additional examples. Moreover, as an application of our approach, we will study reactive systems described by means of transition systems as components and of the usual synchronous product as architectural connector, and whose behavior is expressed by logical properties over a modal first-order logic. In this framework, we propose to study complexity of reactive systems through this notion of emergent properties. We will also give some conditions to guarantee when a system is lacking of non-conformity properties which have been recognized as being the cause of bad interaction between components. This last work has been published in the proceedings of [2]. Here, this manuscript also extends the paper published in [2] with complete proofs of the main results.

2 Institutions

The theory of institutions [20] is a categorical abstract model theory which formalizes the intuitive notion of logical system, including syntax, semantics, and the satisfaction between them. This emerged in computing science studies of software specification and semantics, in the context of the increasing number of considered logics, with the ambition of doing as much as possible at the level of abstraction independent of commitment to any particular logic. Now institutions have become a common tool in the area of formal specification, in fact its most fundamental mathematical structure.

2.1 Basic definitions

Definition. 1 (Institution) *An* institution $\mathcal{I} = (Sig, Sen, Mod, \models)$ *consists of*

- *a category* Sig*, objects of which are called* signatures,
- *a functor* $Sen : Sig \rightarrow Set$ *giving for each signature a set, elements of which are called* sentences,
- *a contravariant functor* $Mod : Sig^{op} \rightarrow Cat$ *giving for each signature a category, objects and arrows of which are called* Σ-models *and* Σ*-morphisms respectively, and*
- *a* $|Sig|$*-indexed family of relations*

$$\models_\Sigma \subseteq |Mod(\Sigma)| \times Sen(\Sigma)$$

called satisfaction relation,

such that the following property holds:
$\forall \sigma : \Sigma \rightarrow \Sigma',\ \forall \mathcal{M}' \in |Mod(\Sigma')|,\ \forall \varphi \in Sen(\Sigma)$,

$$\mathcal{M}' \models_{\Sigma'} Sen(\sigma)(\varphi) \Leftrightarrow Mod(\sigma)(\mathcal{M}') \models_\Sigma \varphi$$

Here, we define some notions over institutions which will be useful thereafter.

Definition. 2 (Elementary equivalence) *Let* $\mathcal{I} = (Sig, Sen, Mod, \models)$ *be an institution. Let* Σ *be a signature. Two* Σ*-models* M_1 *and* M_2 *are* elementary equivalent*, noted* $M_1 \equiv_\Sigma M_2$ *if, and only if the following condition holds:* $\forall \varphi \in Sen(\Sigma),\ M_1 \models_\Sigma \varphi \Longleftrightarrow M_2 \models_\Sigma \varphi$.

This means that $\mathcal{M}_1$ and $\mathcal{M}_2$ are undistinguishable with respect to the formula satisfaction.

Definition. 3 (Closed under isomorphism) *An institution is* closed under isomorphism *if, and only if every two isomorphic models are elementary equivalent.*

All reasonable logics (anyway all the logics classically used in mathematics and computer science) are closed under isomorphism.

Definition. 4 (Logical theory) *Let $\mathcal{I} = (Sig, Sen, Mod, \models)$ be an institution. Let Σ be a signature of $|Sig|$. Let T be a set of Σ-sentences (i.e. $T \subseteq Sen(\Sigma)$). Let us denote $Mod(T)$ the full sub-category of $Mod(\Sigma)$ whose objects are all Σ-models $\mathcal{M}$ such that for any $\varphi \in T$, $\mathcal{M} \models_\Sigma \varphi$, and $T^\bullet$ the subset of $Sen(\Sigma)$, so-called* semantic consequences of T*, defined as follows:*

$$T^\bullet = \{\varphi \mid \forall \mathcal{M} \in |Mod(T)|,\ \mathcal{M} \models_\Sigma \varphi\}$$

T is a logical theory *if, and only if $T = T^\bullet$.*

$\varphi \in T^\bullet$ is also denoted by $T \models_\Sigma \varphi$.

2.2 Examples of institutions

2.2.1 Propositional Logic (PL)

Signatures and signature morphisms are sets of propositional variables and functions between them respectively.
Given a signature Σ, the set of Σ-sentences is the least set of sentences finitely built over propositional variables in Σ and Boolean connectives in $\{\neg, \vee, \wedge, \Rightarrow\}$. Given a signature morphism $\sigma : \Sigma \to \Sigma'$ associating to each propositional variable of Σ a propositional variable of Σ', $Sen(\sigma)$ translates Σ-formulas to Σ'-formulas by renaming propositional variables according to σ.
Given a signature Σ, the category of Σ-models is the category of mappings[1] $\nu : \Sigma \to \{0, 1\}$ with identities as morphisms. Given a signature morphism $\sigma : \Sigma \to \Sigma'$, the forgetful functor $Mod(\sigma)$ maps a Σ'-model ν' to the Σ-model $\nu = \nu' \circ \sigma$.
Finally, satisfaction is the usual propositional satisfaction.

2.2.2 Many-sorted First Order Logic with equality (FOL)

Signatures are triples (S, F, P) where S is a set of sorts, and F and P are sets of function and predicate names respectively, both with arities in $S^* \times S$ and S^+ respectively.[2]
Signature morphisms $\sigma : (S, F, P) \to (S', F', P')$ consist of three functions between sets of sorts, sets of functions and sets of predicates respectively, the last two preserving arities.
Given a signature $\Sigma = (S, F, P)$, the Σ-atoms are of two possible forms: $t_1 = t_2$ where [3] $t_1, t_2 \in T_F(X)_s$ ($s \in S$), and $p(t_1, \ldots, t_n)$ where $p : s_1 \times \ldots \times s_n \in P$ and $t_i \in T_F(X)_{s_i}$ ($1 \leq i \leq n$, $s_i \in S$). The set of Σ-sentences is the least set of formulas built over the set of Σ-atoms by finitely applying Boolean connectives in $\{\neg, \vee, \wedge, \Rightarrow\}$ and quantifiers $\forall$ and $\exists$.
Given a signature $\Sigma = (S, F, P)$, a Σ-model $\mathcal{M}$ is a family $M = (M_s)_{s \in S}$ of sets (one for every $s \in S$), each one equipped with a function $f^\mathcal{M} : M_{s_1} \times \ldots \times M_{s_n} \to M_s$ for every $f : s_1 \times \ldots \times s_n \to s \in F$ and with a n-ary relation $p^\mathcal{M} \subseteq M_{s_1} \times \ldots \times M_{s_n}$ for every $p : s_1 \times \ldots \times s_n \in P$.
Given a signature morphism $\sigma : \Sigma = (S, F, P) \to \Sigma' = (S', F', P')$ and a Σ'-model $\mathcal{M}'$, $Mod(\sigma)(\mathcal{M}')$ is the Σ-model $\mathcal{M}$ defined for every $s \in S$ by $M_s = M'_{\sigma(s)}$, and for every function name $f \in F$ and predicate name $p \in P$, by $f^\mathcal{M} = \sigma(f)^{\mathcal{M}'}$ and $p^\mathcal{M} = \sigma(p)^{\mathcal{M}'}$. Finally, satisfaction is the usual first-order satisfaction.

Many other important logics can be obtained as FOL restrictions such as:

- **Horn Clause Logic (HCL).** An *universal Horn sentence* for a signature Σ in **FOL** is a Σ-sentence of the form $\Gamma \Rightarrow \alpha$ where Γ is a finite conjunction of Σ-atoms and α is a Σ-atom. The institution of Horn clause logic is the sub-institution of **FOL** whose signatures and models are those of **FOL** and sentences are restricted to the universal Horn sentences.

- **Equational Logic (EQL).** An *algebraic signature* (S, F) simply is a **FOL** signature without predicate symbols. The institution of equational logic is the sub-institution of **FOL** whose signatures and models are algebraic signatures and algebras respectively.

- **Conditional equational logic (CEL).** The institution of conditional equational logic is the sub-institution of **EQL** whose sentences are universal Horn clauses for algebraic signatures.

- **Rewriting Logic (RWL)** Given an algebraic signature $\Sigma = (S, F)$, Σ-sentences are formulas of the form $\varphi : t_1 \to t'_1 \wedge \ldots \wedge t_n \to t'_n \Rightarrow t \to t'$ where $t_i, t'_i \in T_F(X)_{s_i}$ ($1 \leq i \leq n$, $s_i \in S$) and $t, t' \in T_F(X)_s$ ($s \in S$). Models of rewriting logic are preorder models, *i.e.* given a signature $\Sigma = (S, F)$, $Mod(\Sigma)$ is the category of Σ-algebras $\mathcal{A}$ such that for every $s \in S$, A_s is equipped with a preorder $\geq$. Hence, $\mathcal{A} \models \varphi$ if, and only if for every variable assignment $\nu : X \to A$, if each $\nu(t_i)^\mathcal{A} \geq \nu(t'_i)^\mathcal{A}$ then $\nu(t)^\mathcal{A} \geq \nu(t')^\mathcal{A}$ where $\_^A : T_F(A) \to A$ is the mapping inductively defined by: $f(t_1, \ldots, t_n)^A = f^\mathcal{A}(t_1^A, \ldots, t_n^A)$.

2.2.3 Modal FOL (MFOL)

Signatures are couples (Σ, A) where Σ is a **FOL**-signature and A is a set of actions, and morphisms are couples of **FOL**-signature morphisms and total functions on sets of actions. In the sequel, we will note by the same name both **MFOL**-signature and each of its components.

[1] $\{0, 1\}$ are the usual truth-values.
[2] S^+ is the set of all non-empty sequences of elements in S and $S^* = S^+ \cup \{\epsilon\}$ where ϵ denotes the empty sequence.
[3] $T_F(X)_s$ is the term algebra of sort s built over F with sorted variables in a given set X.

Given a **MFOL** signature (Σ, A) with $\Sigma = (S, F, P)$, (Σ, A)-atoms are either predicates $p(t_1, \ldots, t_n)$ or the symbol T (for True), and the set of (Σ, A)-formulas is the least set of formulas built over the set of (Σ, A)-atoms by finitely applying Boolean connectives in $\{\neg, \vee, \wedge, \Rightarrow\}$, quantifiers $\forall$ and $\exists$, and modalities in $\{\Box_a | a \in A\}$. For every $a \in A$, the intuitive meaning of $\Box_a$ is "always after the action a". Given a signature (Σ, A), a (Σ, A)-model (W, R), called Kripke frame, consists of a family $W = (W^i)_{i \in I}$ of Σ-models in **FOL** (the *possible worlds*) such that [4] $W^i_s = W^j_s$ for every $i, j \in I$ and $s \in S$, and a A-indexed family of "accessibility" relations $R_a \subseteq I \times I$. Given a signature morphism $\sigma : (\Sigma, A) \rightarrow (\Sigma', A')$ and a (Σ', A')-model $((W'^i)_{i \in I}, R')$, $Mod(\sigma)(((W'^i)_{i \in I}, R'))$ is the (Σ, A)-model $(Mod(\sigma)(W'^i)_{i \in I}, R)$ defined for every $a \in A$ by $R_a = R'_{\sigma(a)}$. A (Σ, A)-sentence φ is said to be satisfied by a (Σ, A)-model (W, R), noted $(W, R) \models_{(\Sigma, A)} \varphi$, if for every $i \in I$ we have $(W, R) \models^i_\Sigma \varphi$, where $\models^i_\Sigma$ is inductively defined on the structure of φ as follows:

- for every **FOL**-formula φ built over Σ-atoms, $(W, R) \models^i_\Sigma \varphi$ iff $W^i \models_\Sigma \varphi$
- $(W, R) \models^i_\Sigma \Box_a \varphi$ when $(W, R) \models^j_\Sigma \varphi$ for every $j \in I$ such that $i \; R_a \; j$.

2.2.4 More exotic institutions

The institution theory also enables to represent formalisms which are not logics strictly speaking.

Formal languages (FL) The institution of formal languages is defined by the category of signatures Set. Given a set A, the set of sentences is A^* and $Mod(A)$ is the category whose objects are all subsets of A^*. Given a signature morphism $\sigma : A \rightarrow A'$, $Mod(\sigma)$ is the functor which at $L' \subseteq A'^*$ associates the set $L = \{\alpha | \sigma(\alpha) \in L'\}$. Finally, given a signature $\Sigma \in Sig$, $\models_\Sigma$ is just the membership relation $\ni$. It is obvious to show that the satisfaction condition holds.

Programming languages (PLG) The institution of a programming language [28] is built over an algebra of built-in data types and operations of a programming language. Signatures are FOL signatures and sentences are programs of the programming language over signatures; and models are algebraic structures in which functions are interpreted as recursive mappings (i.e for each function symbol is assigned a computation (either diverging, or yielding a result) to any sequence of actual parameters). A model satisfies a sentence if, and only if it assigns to each sequence of parameters the computation of the function body as given by the sentence. Hence, sentences determine particular functions in the model uniquely. Finally, signature morphisms, model reductions and sentence translations are defined similarly to those in FOL.

[4] In the literature, Kripke frames satisfying such a property are said *with constant domains*.

3 Specifications in institutions

Over institutions, specifications are usually defined either by logical theories or couples (Σ, Ax) where Σ is a signature and Ax a set (usually finite) of formulae (often called axioms) over Σ. However, there is a large family of specification formalisms mainly used to specify concurrent, reactive and dynamic systems for which specifications are not expressed in this way. We can cite for instance process algebras, transition systems or Petri nets. Now, all of these kinds of specifications can be studied through the set of their semantic consequences expressed in an adequate formalism. This leads us up to define the notion of specifications over institutions.

3.1 Definitions

Let us now consider a fixed but arbitrary institution $\mathcal{I} = (Sig, Sen, Mod, \models)$.

Definition. 5 (Specifications) *A* specification language $\mathcal{SL}$ *over* $\mathcal{I}$ *is a pair* $(Spec, Real)$ *where:*

- $Spec : Sig^{op} \rightarrow Set$ *is a functor. Given a signature* Σ*, elements in* $Spec(\Sigma)$ *are called* specifications over Σ.
- $Real = (Real_\Sigma)_{\Sigma \in |Sig|}$ *is a* Sig*-indexed family of mappings* $Real_\Sigma : Spec(\Sigma) \rightarrow |Cat|$ *such that for every* $\Sigma \in |Sig|$*, and every* $Sp \in Spec(\Sigma)$*,* $Real_\Sigma(Sp)$ *is a full subcategory of* $Mod(\Sigma)$*. Objects of* $Real_\Sigma(Sp)$ *are called* realizations of Sp.

Definition. 6 (Semantic consequences) *Let* $\mathcal{SL} = (Spec, Real)$ *be a specification language over* $\mathcal{I}$*. Let us define* $\_^\bullet = (\_^\bullet_\Sigma)_{\Sigma \in Sig}$ *the* Sig*-indexed family of mappings* $\_^\bullet_\Sigma : Spec(\Sigma) \rightarrow \mathcal{P}(Sen(\Sigma))$ *that to every* $Sp \in Spec(\Sigma)$*, yields the set* $Sp^\bullet_\Sigma = \{\varphi | \forall \mathcal{M} \in Real_\Sigma(Sp), \mathcal{M} \models_\Sigma \varphi\}$*.* $Sp^\bullet_\Sigma$ *is called the* set of semantic consequences *of* Sp *or the* theory of Sp.

Definition 6 calls for some comments:

- We could expect that $Mod(Sp^\bullet) = Real(Sp)$ what would make unmeaning the existence of the mappings in $Real$ in Definition 5. However, we can often be led up to make some restrictions on specification models. For instance, when dealing with axiom specifications expressed in equational logic, we can be interested by reachable or initial models to allow inductive proofs or for computability reasons.

- Sometimes, $\_^\bullet$ is a natural transformation from $Spec$ to[5] $\mathcal{P} \circ Sen^{op}$. However, most of times, it is not the case (see the examples in Section 3.2).

Definition. 7 (Category of specifications) *Let $\mathcal{SL}$ be a specification language over an institution $\mathcal{I}$. Denote $SPEC$ the category of specifications over $\mathcal{SL}$ whose the objects are the elements in $\bigcup_{\Sigma \in |Sig|} Spec(\Sigma)$, and morphisms are actually given by signature morphisms (i.e. for every $Sp \in Spec(\Sigma)$ and every $Sp' \in Spec(\Sigma')$, $\sigma : Sp \to Sp' \in SPEC$ iff $\sigma : \Sigma \to \Sigma' \in Sig$). If a morphism $\sigma : Sp \to Sp'$ in $SPEC$ further satisfies: $Sen(\sigma)(Sp^\bullet_\Sigma) \subseteq Sp'^\bullet_{\Sigma'}$, then σ is called* specification morphism.
$Sig : SPEC \to Sig$ is the functor which maps any specification $Sp \in Spec(\Sigma)$ to the signature Σ and any morphism σ to the signature morphism $Sig(\sigma)$.

Hence, specification morphisms are arrows in $SPEC$ that further preserve semantic consequences. Commonly, the category of specifications over institutions have $\bigcup_{\Sigma \in |Sig|} Spec(\Sigma)$ as objects and specification morphisms as arrows [20, 29]. Here, the fact to consider just signature morphisms between specifications will be useful to define both architectural connectors and their combination.

3.2 Examples of specifications

We give three examples of specification languages that correspond to the usual forms of specifications over arbitrary institutions.

3.2.1 Logical theories

Here, specifications are logical theories. To meet the requirements given in Definition 5, this gives rise to the functor $Spec : Sig^{op} \to Set$ which to every $\Sigma \in Sig$, associates the set of all Σ-theories T, and to every signature morphism $\sigma : \Sigma \to \Sigma'$, matches every Σ'-theory T' with the Σ-theory $T = \{\varphi | Sen(\sigma)(\varphi) \in T'\}$. Hence, $Spec(\Sigma) \subseteq \mathcal{P}(Sen(\Sigma))$. We naturally define $Real_\Sigma(T) = Mod(T)$. Moreover, specifications being saturated theories, this naturally leads to the identity function $\_^\bullet_\Sigma : Spec(\Sigma) \to \mathcal{P}(Sen(\Sigma))$. It is easy to check that given a signature morphism $\sigma : \Sigma \to \Sigma'$, the following diagram commutes and then $\_^\bullet$ is a natural transformation:

$$\begin{array}{ccc} Spec(\Sigma) & \xrightarrow{-^\bullet_\Sigma} & \mathcal{P}(Sen(\Sigma)) \\ \uparrow{\scriptstyle Spec(\sigma)} & & \uparrow{\scriptstyle \mathcal{P}(Sen^{op}(\sigma^*))} \\ Spec(\Sigma') & \xrightarrow{-^\bullet_{\Sigma'}} & \mathcal{P}(Sen(\Sigma')) \end{array}$$

(See Footnote 5 for the definition of σ^*)

3.2.2 Axiomatic specifications

In this case, specifications are defined by pairs (Σ, Ax) where Σ is a signature and $Ax \subseteq Sen(\Sigma)$, and given a signature morphism $\sigma : \Sigma \to \Sigma'$, $Spec(\sigma)$ matches every Σ'-specification $Sp' = (\Sigma', Ax')$ to $Sp = (\Sigma, \{\varphi | Sen(\sigma)(\varphi) \in Ax'\})$. By the satisfaction condition, we have that $Sen(\sigma)(Ax^\bullet) \subseteq Ax'^\bullet$. The functor $Spec$ then associates to every signature Σ the set of pairs (Σ, Ax), and $(\Sigma, Ax)^\bullet_\Sigma = Ax^\bullet$. Observe that $\_^\bullet$ is not a natural transformation. Indeed, let us set in **FOL**, and consider the inclusion morphism $\sigma : \Sigma \to \Sigma'$ where $\Sigma' = (\{s\}, \emptyset, \{R_1, R_2 : s \times s\})$ and $\Sigma = (\{s\}, \emptyset, \{R_1 : s \times s\})$. Let Ax' be the set of axioms:

$$x\ R_2\ y \Longrightarrow y\ R_2\ x$$
$$x\ R_1\ y \Longleftrightarrow x\ R_2\ y$$

Obviously, we prove from Ax' that R_1 is a symmetric relation.
However, $Spec(\sigma)((\Sigma', Ax')) = \emptyset$, and then $Spec(\sigma)((\Sigma', Ax'))^\bullet$ is restricted to tautologies while $\mathcal{P}(Sen^{op}(\sigma^*))(Ax')$ contains at least $x\ R_1\ y \Rightarrow y\ R_1\ x$.

3.2.3 Inference rules

In the framework of formal language, languages L over an alphabet A can be specified by inference rules, that is n-ary relations r on A^* and a tuple $(\alpha_1, \ldots, \alpha_n) \in r$ means that if $\alpha_1, \ldots, \alpha_{n-1}$ are words of the language, then so is α_n. Hence, a specification over an alphabet A is a set R of n-ary relations on A^*. Given a signature morphism $\sigma : A \to A'$ and a specification R' over A', the specification $Spec(\sigma)(R')$ over A is the set R of n-ary relation r such that there exists $r' \in R'$ and $r = \{(a_1, \ldots, a_n) | (\forall i, 1 \leq i \leq n, a_i \in A) \wedge (a_1, \ldots, a_n) \in r'\}$. Given a set of inference rules R over an alphabet A, $R^\bullet_A$ is the language L inductively generated from inference rules of R. Given a signature morphism $\sigma : A \to A'$ and a set of inference rules R' over A'. It is easy to show that $Spec(\sigma)(R')^\bullet_A = R'^\bullet_{A'} \cap A^*$ what proves that $\_^\bullet$ is a natural transformation from $Spec$ to $\mathcal{P} \circ Sen^{op}$.

[5] Given a functor $F : \mathcal{C} \to \mathcal{D}$, $\mathcal{F}^{op} : \mathcal{C}^{op} \to \mathcal{D}^{op}$ is the dual of $\mathcal{F}$ defined as follows:

- $\forall o \in \mathcal{C}, F^{op}(o) = F(o)$
- f^* being the reverse arrow of f in $\mathcal{C}$, $\forall o, o' \in \mathcal{C}, \forall f \in Hom_\mathcal{C}(o, o'), F^{op}(f^*) = F(f)^*$

The *powerset functor* $\mathcal{P} : Set^{op} \to Set$ takes a set S to its powerset $\mathcal{P}(S)$, and a set function $f : S \to S'$ (i.e., an arrow from S' to S in Set^{op}) to the inverse image function $f^{-1} : \mathcal{P}(S') \to \mathcal{P}(S)$ which associates to a subset $A \subseteq S'$ the subset $\{s \in S | f(s) \in A\}$ of S.

3.3 Properties of specifications

Proposition. 1 *Let $\sigma : Sp \to Sp'$ be a specification morphism. Then, the functor $Mod(\sigma) : Mod(Sig(Sp')) \to Mod(Sig(Sp))$ can be restricted to specification semantic consequences (i.e. $Mod(\sigma) : Mod(Sp'^{\bullet}_{\Sigma'}) \to Mod(Sp^{\bullet}_{\Sigma})$ is a functor).*

Proof. *Let $\varphi \in Sp^{\bullet}_{Sig(Sp)}$ and $\mathcal{M} \in Mod(Sp')$. As σ is a specification morphism, $\mathcal{M} \models_{Sig(Sp')} Sen(\sigma)(\varphi)$. Therefore, by the satisfaction condition, we also have that $Mod(\sigma)(\mathcal{M}) \models_{Sig(Sp)} \varphi$.*

We cannot state a similar result for the family of mappings $Real$, i.e. we cannot define in a general way a functor of the form $Real(\sigma) : Real(Sp') \to Real(Sp)$. The following notion of compatibility captures the existence of such a functor.

Definition. 8 (Compatible) *Let $\mathcal{SL} = (Spec, Real)$ be a specification language over $\mathcal{I}$. Let $\sigma : Sp \to Sp'$ be a specification morphism. $Real$ is said* compatible *with σ if, and only if we can define a functor $Real(\sigma) : Real(Sp') \to Real(Sp)$.*

Here, we define two other notions that we will use afterwards.

Definition. 9 (Definable by specification) *Given an institution $\mathcal{I}$ and a specification language over $\mathcal{I}$, a Σ-theory T is said* definable by specification *or* definable *for being shorter if, and only if there exists $Sp \in Spec(\Sigma)$ such that $T = Sp^{\bullet}_{\Sigma}$.*

In the following definition, we now adapt the standard notion of liberal specification morphism [12] which will be useful in Section 4.3.

Definition. 10 (Liberality) *In any specification language $\mathcal{SL}$ over $\mathcal{I}$, a specification morphism $\sigma : Sp \to Sp'$ is* liberal *if, and only if $Real$ is compatible with σ and $Real(\sigma) : Real(Sp') \to Real(Sp)$ has a left-adjunct $\mathcal{F}(\sigma) : Real(Sp) \to Real(Sp')$.*

Specifications defined by logical theories and axiomatic specifications over the institution **CEL** is liberal for every specification morphism σ. Indeed, let $\sigma : \Sigma = (S, F) \to \Sigma' = (S', F')$ be a signature morphism, and let Γ and Γ' be two sets of conditional equations over, respectively, Σ and Σ' such that $Sen(\sigma)(\Gamma) \subseteq \Gamma'$. We can build a functor $T_{\Gamma'/\Gamma} : \mathcal{A} \mapsto T_{\Gamma'/\Gamma}(\mathcal{A})$, from the category of Γ-algebras to the category of Γ'-algebras.
Let $\mathcal{A}$ be a Γ-algebras. $T_{\Gamma'/\Gamma}(A)$ is the quotient of $T_{F'}(\mathcal{A})$ by the congruence generated by the kernel of the Σ-morphism $T_F(A)$ in $\mathcal{A}$ extending the identity on X. [6] This algebra satisfies the following universal property: for every Γ'-algebra $\mathcal{B}$ and every Σ-morphism $\mu : \mathcal{A} \to Mod(\sigma)\mathcal{B}$, there exists a unique Σ'-morphism $\eta_{\mathcal{B}} : T_{\Gamma'/\Gamma}(\mathcal{A}) \to \mathcal{B}$ such that for every $a \in A$, $\eta_{\mathcal{B}}(a) = \mu(a)$. This universal property directly shows that the functor $T_{\Gamma'/\Gamma}$ is left-adjunct to $Mod(\sigma)$, i.e., for every Γ-algebra $\mathcal{A}$ there exists a universal morphism $\mu_{\mathcal{A}} : \mathcal{A} \to Mod(\sigma)(T_{\Gamma'/\Gamma}(\mathcal{A}))$. $\mu_{\mathcal{A}}$ is called the *adjunct morphism* for $\mathcal{A}$.

4 Architectural connector

4.1 Definitions

Succinctly, architectural connectors enable one to combine components (specifications) together to make bigger ones. However, depending on the used specification language, the way of combining components can be different. For instance, when specifications are logical theories then their combination is often based on the set theoretical union on signatures whereas the combination of specifications made of transition systems is based on some kinds of product. However, one can observe that most of existing connectors c have the following common features:

- a connector c gets as arguments a fixed number n of existing specifications Sp_1, Sp_2, $\ldots Sp_n$ defined respectively over the signatures Σ_1, Σ_2, $\ldots \Sigma_n$, to build a new one, denoted $Sp = c(Sp_1, Sp_2, \ldots, Sp_n)$. We can then see the connector c as a mapping of arity n from $|SPEC|^n$ to $|SPEC|$. We will see in the examples that actually c may be a partial function, but often defined in a way sufficiently general to accept as arguments tuples $(Sp_1, Sp_2, \ldots, Sp_n)$ with a large associated family of signature tuples $(\Sigma_1, \Sigma_2, \ldots, \Sigma_n)$.

- as specifications will be recursively defined by means of connectors, the arguments Sp_1, Sp_2, $\ldots Sp_n$ of the connector c can be linked together by some constraints on elements present in specification signatures, expressed by signature morphisms. These constraints will be taken into account by the definition of the connector c. Hence, the arguments of a connector c will not be a tuple of n specifications, but n specifications equipped with signature morphisms. This will be defined by a graph whose nodes are specifications and edges are signature morphisms. In our category theory based setting, such a graph is called a diagram of the specification category $SPEC$. In practice, for a given connector c, all the diagrams accepted as arguments by

[6] $T_{F'}(A)$ (resp. $T_F(A)$) is the term algebra built over F' (resp. F) with sorted variables in the carrier A of the Γ-algebra $\mathcal{A}$.

c have the same graph shape (i.e. the same organization between nodes and edges). Hence, our connectors will be built on the diagram category with the same shape over the category $SPEC$.

- the signature Σ of Sp is the least one over the signatures Σ_1, Σ_2, ..., Σ_n. This expresses the fact that generally, a connector c does not explicitly introduce new elements to be specified, but on the contrary only combines the elements already present in one of the signatures Σ_1, Σ_2 ... Σ_n. In the following definition of connectors, this will be expressed by the co-limit of the diagram, projected on signatures.

This then leads us up to formally define architectural connectors as follows:

Notation. 1 (Diagram category) *Let I and C be two categories. Note $\Delta_{(I,C)}$ the category of diagrams in C with shape I, i.e. the category whose objects are all functors $\delta : I \to C$, and morphisms are natural transformations between functors $\delta, \delta' : I \to C$.*
Let I' be a subcategory of a category I. Let δ be a diagram of $\Delta_{(I,C)}$. Let us denote $\delta_{|_{I'}}$ the diagram of $\Delta_{(I',C)}$ obtained by restricting δ to I'.

Definition. 11 (Co-cone) *Given a diagram $\delta : I \to C$. A* co-cone *of δ consists of an object $c \in |C|$ and a I-indexed family of morphisms $\alpha_i : \delta(i) \to c$ such that for each edge $e : i \to i'$ in I, we have that $\alpha_{i'} \circ \delta(e) = \alpha_i$.*

A co-limiting co-cone (co-limit) $(c, \{\alpha_i\}_{i \in I})$ can be understood as a minimal co-cone, that is:

Definition. 12 (Co-limit) *A co-cone $(c, \{\alpha_i\}_{i \in I})$ of a diagram δ is a* co-limit *if, and only if it has the property that for any other co-cone $(d, \{\beta_i\}_{i \in I})$ of δ, there exists a unique morphism $\gamma : c \to d$ such that for every $i \in I$, $\gamma \circ \alpha_i = \beta_i$. When I is the category $\bullet \leftarrow \bullet \rightarrow \bullet$ with three objects and two non-identity arrows, the co-limit is called a* pushout.

Definition. 13 (Co-complete) *A category C is* co-complete *if for every shape category I, every diagram $\delta : I \to C$ has a co-limit.*

In the sequel, we will then consider institutions whose the signature category is co-complete.

Definition. 14 (Architectural connector) *Let $\mathcal{SL}$ be a specification language over an institution $\mathcal{I}$ for which the category Sig is co-complete. An* architectural connector *$c : |\Delta_{(I,SPEC)}| \to |SPEC|$ is a partial mapping such that every $\delta \in \Delta_{(I,SPEC)}$ for which $c(\delta)$ is defined, is equipped with a co-cone $p : Sig \circ \delta \to Sig(c(\delta))$ co-limit of $Sig \circ \delta$.*

Example. 1 (Enrichment and union) *Enrichment and union of specifications have surely been the first primitives architectural connectors (so-called structuring primitives) to be formally defined and studied especially when dealing with specifications defined as axiomatic specifications. They even received an abstract formalization in institutions [8]. In our framework, both structuring primitives are defined as follows: we consider an institution $\mathcal{I} = (Sig, Sen, Mod, \models)$. Moreover, in Example 1, $SPEC$ is the category whose objects are specifications of the form (Σ, Ax) over a given institution $\mathcal{I}$ and morphisms are any $\sigma : (\Sigma, Ax) \to (\Sigma', Ax')$ s.t. $\sigma : \Sigma \to \Sigma'$ is a signature morphism.*

Enrichment *Let I be the graph composed of two nodes i and j and one arrow $a : i \to j$. The connector* Enrich *for axiomatic specifications is defined for every diagram $\delta : I \to SPEC$ where $\delta(i) = (\Sigma, Ax)$ and $\delta(j) = (\Sigma', Ax')$ such that $Sen(Sig(\delta(a)))(Ax) \subseteq Ax'$, and yields $Enrich(\delta) = (\Sigma', Ax')$ together with the co-cone $Sig(\delta(a))$ and $Id_{Sig(\delta(j))}$ which is the obvious co-limit of $Sig \circ \delta$. Observe that $\delta(a)$ and $Id_{\delta(j)}$ are further specification morphisms.*

Union *Let I be the graph composed of three nodes i, j, and k and two arrows $a_1 : i \to j$ and $a_2 : i \to k$. The connector $Union$ is defined for every diagram $\delta : I \to SPEC$ where $\delta(i) = (\Sigma_0, Ax_0)$, $\delta(j) = (\Sigma_1, Ax_1)$ and $\delta(k) = (\Sigma_2, Ax_2)$, and such that $Sen(Sig(\delta(a_1)))(Ax_0) \subseteq Ax_1$ and $Sen(Sig(\delta(a_2)))(Ax_0) \subseteq Ax_2$, and yields $Union(\delta) = (\Sigma, Ax)$ with the co-cone $p : Sig \circ \delta \to \Sigma$ which is the pushout of $Sig(\delta(a_1))$ and $Sig(\delta(a_2))$ and such that $Ax = Sen(p_j)(Ax_1) \cup Sen(p_k)(Ax_2)$. Observe that we can derive the co-cone $p_{SPEC} : \delta \to (\Sigma, Ax)$ such that $Sig \circ p_{SPEC} = p$, and p_{SPEC_j} and p_{SPEC_k} are specification morphisms.*

In [8], both above connectors have been brought down to two basic connectors: union with constant signatures $\bigcup$, and **translate _ by** *σ for every signature morphism σ. They are defined by:*

1. *Let I be the graph composed of two nodes i and j and without arrows between i and j. The connector $\bigcup$ is defined for every diagram $\delta : I \to SPEC$ where $\delta(i) = (\Sigma, Ax_1)$ and $\delta(j) = (\Sigma, Ax_2)$, and yields $\bigcup(\delta) = (\Sigma, Ax)$ with the obvious co-limit $p : Sig \circ \delta \to \Sigma$ where p_i and p_j are the identity signature morphism for Σ, and such that $Ax = Ax_1 \cup Ax_2$.*

2. *Let I be the graph composed two nodes k and l. The connector* **translate _ by** *σ where $\sigma : \Sigma \to \Sigma'$ is a signature morphism, is defined for every diagram $\delta : I \to SPEC$ where $\delta(k) = (\Sigma, Ax)$ and $\delta(l) = (\Sigma', Sen(\sigma)(Ax))$, and yields* **translate _ by** *$\sigma(\delta) = (\Sigma', Sen(\sigma)(Ax))$.*

In [8], $\bigcup(\delta)$ *and* **translate _ by** $\sigma(\delta)$ *are respectively noted* $\delta(i) \bigcup \delta(j)$ *and* **translate** $\delta(k)$ **by** σ.

Architectural connectors can be combined to deal with specifications in the large.

Definition. 15 (Connector combination) *Let* $c : |\Delta_{I,SPEC}| \rightarrow |SPEC|$ *and* $c' : |\Delta_{(I',SPEC)}| \rightarrow |SPEC|$ *be two architectural connectors. Let* $i' \in |I'|$ *be an object. Let* $I' \circ_{i'} I$ *be the category defined by:*

- $|I' \circ_{i'} I| = |I| \coprod |I'|$
- *the sets* $Hom_{I' \circ_{i'} I}(k,l)$ *for every* $k, l \in |I' \circ_{i'} I|$ *are inductively defined as follows:*
  - $k, l \in |I'| \Rightarrow Hom_{I'}(k,l) \subseteq Hom_{I' \circ_{i'} I}(k,l)$
  - $k, l \in |I| \Rightarrow Hom_{I}(k,l) \subseteq Hom_{I' \circ_{i'} I}(k,l)$
  - *for every* $i \in |I|$*, we introduce the arrow* q_i *in* $Hom_{I' \circ_{i'} I}(i, i')$.
  - $Hom_{I' \circ_{i'} I}$ *is closed under composition.*

Let us denote $c' \circ_{i'} c : |\Delta_{I' \circ_{i'} I, SPEC}| \rightarrow |SPEC|$ *the architectural connector defined by :* [7]

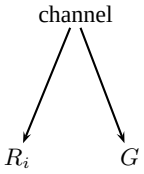
$$\delta \mapsto \begin{cases} c'(\delta_{|_{I'}}) & \text{if } c(\delta_{|_I}) \text{ is defined} \\ & \delta(i') = c(\delta_{|_I}) \\ & \text{and } \delta(q_i) \text{ is the morphism } r_i \text{ in } SPEC \\ & \text{whose the image by } Sig \text{ is the component} \\ & p_i \text{ of the co-limit } p \text{ associated to } c(\delta_{|_I}) \\ undefined & otherwise \end{cases}$$

Example. 2 *Enrichment can be removed and replaced by the following combination of* **translate** *and* $\cup$ *as follows: let* δ *be a diagram of* $\Delta_{(I,SPEC)}$ *where* I *is the index category of the connector* $Enrich$*,* $\delta(i) = (\Sigma, Ax)$ *and* $\delta(j) = (\Sigma', Ax')$

$$Enrich(\delta) = \bigcup \circ_i \textbf{translate\_by} \delta'(p_i)(\delta')$$

where δ' *is the diagram of* $\Delta_{(I'' \circ_i I', SPEC)}$ *for* I'' *(resp.* I'*) the index category of the connector* $\cup$ *(resp.* **translate***), defined by:* $\delta'(k) = \delta(i)$*,* $\delta'(i) =$ **translate** $\delta'(k)$ **by** $\delta'(p_i) = (\Sigma', Sen(Sig(p_i))(Ax))$ *and* $\delta'(j) = (\Sigma', Ax' \setminus Ax)$.

The reader accommodated to the terminology and to the concepts of software architecture can be disappointed by the way connectors are interpreted here, i.e. by functions that take components and produce systems. Indeed, connectors are typically viewed as forms of communicating components. Such connectors can also be formalized in our framework. For instance, in Community [15, 16], in the style of Allen and Garlan [6], a connector consists of n roles R_i and one glue G stating the interaction between roles (i.e. the way roles communicate together). Roles and glue are programs defined over signatures (see [16] for a complete definition of programs). In our framework, programs denote specifications from which we can observe temporal properties. Each role and the glue are interconnected by a channel to denote via signature morphisms shared attributes and actions. This gives rise to a diagram defined as the interconnection on the glue G of basic diagrams of the form:



In Community, the mathematical meaning of a connector is then defined by the colimit of such diagrams. This can be easily defined in our framework by considering a connector c defined for every diagram of the previous form over the category $PROG$ (defined in [16]) taken as the category $SPEC$.

4.2 Complex structuring

As already explained in the introduction of the paper, an architectural connector will be considered as complex when:

1. The global system does not preserve the complete behavior of some subsystems. We will then talk about *non-conformity properties*.
2. Some global properties cannot be deduced from a complete knowledge of these components. We will then talk about *true emergent properties*.

This is expressed by comparing the set of semantic consequences of subsystems with the ones of the global system up to signature morphisms.

Definition. 16 (Complex connector) *Let* $c : |\Delta_{(I,SPEC)}| \rightarrow |SPEC|$ *be an architectural connector. Let* δ *be a diagram of* $\Delta_{(I,SPEC)}$ *such that* $c(\delta)$ *is defined.* c *is said* complex for δ *if, and only if one of the two following properties fails:*

1. Conformity.
   $\forall i \in I, \forall \varphi \in Sen(Sig(\delta(i))), \varphi \in \delta(i)^{\bullet}_{Sig(\delta(i))} \iff Sen(p_i)(\varphi) \in c(\delta)^{\bullet}_{Sig(c(\delta))}$

[7] q_i is the arrow introduced in $Hom_{I' \circ_{i'} I}(i, i')$.

2. Non true emergence.

$$\forall \varphi \in c(\delta)^{\bullet}_{Sig(c(\delta))}, \bigcup_{i \in I} Sen(p_i)(\delta(i)^{\bullet}_{Sig(\delta(i))}) \models_{Sig(c(\delta))} \varphi$$

A formula φ that makes fail the equivalence of both Point 1. and Point 2. is called emergent property.
If c is not complex for a diagram δ, then it is said modular.

Example. 3 *Here, we give a very simple example of specifications in which modularity fails. Let* **Nat** *be the specification in* **EQL** *defined as follows:*

Specification of Nat Sorts: $S_{Nat} = \{ nat \}$

Functions : $F_{Nat} =$
$\{0 :\rightarrow nat,$
$succ : nat \rightarrow nat,$
$\_ + \_ : nat \times nat \rightarrow nat \}$

Axioms: $Ax_{Nat} =$
$\{x + 0 = x$
$x + succ(y) = succ(x + y)\}$

Let us us enrich this specification by adding operations and axioms to specify stacks of natural numbers. This leads to the following enrichment:

Sorts: $S_{Stack} = \{ nat, stack \}$

Functions : $F_{Stack} = F_{Nat} \cup$
$\{empty :\rightarrow stack,$
$push : nat \times stack \rightarrow stack,$
$pop : stack \rightarrow stack,$
$top : stack \rightarrow nat,$
$high : stack \rightarrow nat\}$

Axioms: $Ax_{Stack} = Ax_{Nat} \cup$
$\{pop(empty) = empty$
$pop(push(e, P)) = P$
$top(push(e, P)) = e$
$high(push(e, P)) = succ(high(P))\}$

If we suppose that realizations are either the initial model or reachable models [8] *of both specifications, then an example of emergent property is:*

$$\forall x, (x = 0) \vee (\exists y, x = succ(y))$$

This is because $high(empty)$ has not been specified to be equal to 0. On the contrary, if we add this equation in Ax_{Stack}, there is not emergent property anymore.

[8] A model is reachable when any of its values is the result of the evaluation of a ground term.

4.3 Conditions for modularity

As we have explained it in the introduction of this manuscript, complex software systems prevent to check their correctness with respect to their specification step by step by taking the benefit of their recursive structure. This leads to the important consequence that adding any component gives rise to a new systems whose the correctness has to be completely (re)checked. It is then important to study general properties that guarantee when a system is not complex (i.e. modular). This is what we propose to do with the two following results.

Theorem 1 states that showing the non-presence of true emergent properties for a connector c and a diagram δ comes to show that $(\bigcup_{i \in I} Sen(p_i)(\delta(i)^{\bullet}_{Sig(\delta(i))}))^{\bullet}$ is definable by $c(\delta)$.

Theorem. 1 *Let c be an architectural connector and δ be a diagram such that $c(\delta)$ is defined. Then, we have:*

$(\bigcup_{i \in I} Sen(p_i)(\delta(i)^{\bullet}_{Sig(\delta(i))}))^{\bullet}$ is definable by $c(\delta)$ if, and only if the set of true emergent properties is empty and each p_i is a specification morphism.

Proof. *The* only if *part. This obviously results from the fact that $(\bigcup_{i \in I} Sen(p_i)(\delta(i)^{\bullet}_{Sig(\delta(i))}))^{\bullet}$ is definable by $c(\delta)$. Indeed, we have $c(\delta)^{\bullet}_{Sig(c(\delta)} = (\bigcup_{i \in I} Sen(p_i)(\delta(i)^{\bullet}_{Sig(\delta(i))}))^{\bullet}$, that is for every $\varphi \in c(\delta)^{\bullet}_{Sig(c(\delta))}$, we have that $\bigcup_{i \in I} Sen(p_i)(\delta(i)^{\bullet}_{Sig(\delta(i))}) \models_{Sig(c(\delta))} \varphi$.*

The if *part. As each p_i of p is a specification morphism, we have that $(\bigcup_{i \in I} Sen(p_i)(\delta(i)^{\bullet}_{Sig(\delta(i))}))^{\bullet} \subseteq c(\delta)^{\bullet}_{Sig(c(\delta)}$. Moreover, as the set of true emerging properties is empty, we have that $c(\delta)^{\bullet}_{Sig(c(\delta)} \subseteq (\bigcup_{i \in I} Sen(p_i)(\delta(i)^{\bullet}_{Sig(\delta(i))}))^{\bullet}$. Hence, $c(\delta)^{\bullet}_{Sig(c(\delta)} = (\bigcup_{i \in I} Sen(p_i)(\delta(i)^{\bullet}_{Sig(\delta(i))}))^{\bullet}$, and then $(\bigcup_{i \in I} Sen(p_i)(\delta(i)^{\bullet}_{Sig(\delta(i))}))^{\bullet}$ is definable by $c(\delta)$.*

By Theorem 1, the architectural connectors $Enrich$, $Union$, $\bigcup$ and **translate _ by** σ have no true emergence properties for any defined diagram.

As we could expect, modularity is a property which holds for some, but certainly not for all architectural connectors. More surprising, even under the condition that

$(\bigcup_{i\in I} Sen(p_i)(\delta(i)^{\bullet}_{Sig(\delta(i))}))^{\bullet}$ is definable by $c(\delta)$ for a connector c and a diagram δ such that $c(\delta)$ is defined, modularity can fail because of non-conformity properties (see Example 3).
In the next theorem, we give a supplementary condition based on the liberality of each p_i of the co-limit p, that leads to an empty set of non-conformity properties. For Theorem 2, we suppose the following conditions :

1. the institution under consideration is closed under isomorphism,
2. $Real$ is compatible for every specification morphism p_i of the associated co-cone p, and
3. each p_i of the co-limit p associated to the connector c in $\Delta_{(I,SPEC)}$ satisfies the supplementary condition, so-called *Right Satisfaction Condition* (RSC) : $\forall \varphi \in Sen(Sig(\delta_i)), \forall \mathcal{M} \in Real(c(\delta)), Real(p_i)(\mathcal{M}) \models_{Sig(\delta_i)} \varphi \Longrightarrow \mathcal{M} \models_{Sig(\delta(c))} Sen(p_i)(\varphi)$.

The interest of RSC is, realizations being a subset of models, some pruning on realizations in $Real(\delta(c))$ have been allowed to be done, and then this direction of the satisfaction condition has been able to be brought into failure. For instance, this property does not hold when specifications are logical theories and realizations are restricted to reachable models (see Example 3). For the next theorem, we suppose that these three conditions hold.

Theorem. 2 *Let c be an architectural connector and δ be a diagram such that $c(\delta)$ is defined. Suppose that $(\bigcup_{i\in I} Sen(p_i)(\delta(i)^{\bullet}_{Sig(\delta(i))}))^{\bullet}$ is definable by $c(\delta)$, $Real$ is compatible with each p_i and each p_i is liberal. Then, for every $i \in I$ and every $\mathcal{M} \in Real(\delta(i))$, If each adjunct morphism $\mu_{\mathcal{M}} : \mathcal{M} \to Real(p_i)(\mathcal{F}(p_i)(\mathcal{M}))$ is an isomorphism, then the set of non-conformity properties is empty.*

Proof. *Let $\varphi \in \delta(i)^{\bullet}_{Sig(\delta(i))}$, and let $\mathcal{M} \in Real(c(\delta))$. As $(\bigcup_{i\in I} Sen(p_i)(\delta(i)^{\bullet}_{Sig(\delta(i))}))^{\bullet}$ is definable by $c(\delta)$, $Real(p_i)(\mathcal{M}) \models_{Sig(\delta(i))} \varphi$. Therefore, by the hypothesis that the truth of property is preserved for the functor $Real$ through each signature morphism p_i, we have that $\mathcal{M} \models_{Sig(c(\delta))} Sen(p_i)(\varphi)$.*

let $\varphi \in Sen(\delta(i))$ such that $Sen(p_i)(\varphi) \in c(\delta)^{\bullet}$, and let $\mathcal{M} \in Real(\delta(i))$. As $\mathcal{F}(p_i)$ is left-adjunct to $Real(p_i)$, we have $\mathcal{F}(p_i)(\mathcal{M}) \models_{Sig(c(\delta))} Sen(p_i)(\varphi)$. As $Real$ is compatible with each p_i, $Real(\sigma)(\mathcal{F}(p_i)(\mathcal{M})) \models_{Sig(\delta(i))} \varphi$. As the adjunct morphism is an isomorphism and $\mathcal{I}$ is stable under isomorphism, $\mathcal{M}$ and $Real(\sigma)(\mathcal{F}(p_i)(\mathcal{M}))$ are elementary equivalent, and then $\mathcal{M} \models_{Sig(\delta(i))} \varphi$.

Theorem 2 generalizes to any architectural connectors the standard condition of modularity based on the two notions of hierarchical consistency and sufficient completeness [22], which has been stated for the enrichment connector in the algebraic specification framework (when specifications are conditional positive).

5 Application to reactive systems

In this section, we propose to exemplify our abstract framework to reactive system modeling. We will then give a rigorous and formal definition of emergent properties in the framework of reactive system modeling. We restrict ourselves to reactive systems described by means of the usual synchronous product of transition systems, and whose behavior is expressed by logical properties over **MFOL**. The reason is this is sufficient for the purpose of the study, and the results given in this paper could easily be adapted to temporal logics more classically used to reason on reactive systems and other composition connector whose the greatest number are based-on transition system product. In our setting, we will study some conditions under which non-conformity properties do not occur. The interest is this provides guidance in the design process. Indeed, the appearance of non-conformity properties leads to make a posteriori verification of the global system without benefiting from the decomposition of the system into components.

In Section 5.1, we introduce transition systems and their semantics, and define the synchronous product as means to compose them. Finally, Section 5.2 presents results ensuring the non-existence of non-conformity properties along synchronous product.

5.1 Transition systems

5.1.1 Syntax

As usual when considering automata, transition systems describe possible evolutions of system states. Elementary evolutions are represented by a transition relation between states. Each transition between two states is labeled by three elements: actions of the system, guards expressed here by formulas of **FOL** presented in Section 2, and side-effects on states defined by pairs of ground terms or of the form $(p(t_1, \ldots, t_n), b)$ where $p(t_1, \ldots, t_n)$ is a ground atom and b is equal to $true$ or $false$. As usual, we start by defining the language, so-called signature, on which transition systems are built:

Definition. 17 (Signature) *A* signature *is a triple $\mathscr{L} = (\Sigma, V, A)$ where: Σ is a* **FOL***-signature, V is a set of variables over Σ and A is a set whose elements are called* actions.

Definition. 18 (Side-effect) *Given a signature* $\mathscr{L} = (\Sigma, V, A)$ *where* $\Sigma = (S, F, P)$*, a* side-effect *over* $\mathcal{L}$ *is a pair of ground terms over* Σ (t, t') *of the same sort (i.e.* $\exists s \in S, t, t' \in T_F$*) or a couple* $(p(t_1, \ldots, t_n), b)$ *where* $p(t_1, \ldots, t_n)$ *is a ground* Σ*-atom (i.e. each* t_i *is a ground term) and* b *is equal to* $true$ *or* $false$*. In the sequel, a side-effect* (t, t') *will be noted* $t \mapsto t'$*.*
We note $\mathcal{SE}(\mathcal{L})$ *the set of side-effects over* $\mathcal{L}$*.*

A transition system is then defined as follows:

Definition. 19 (Transition system) *Given a signature* $\mathscr{L} = (\Sigma, V, A)$*, a* transition system *is a couple* $(Q, \mathbb{T})$ *where:*

- Q *is a set of* states*, and*
- $\mathbb{T} \subseteq Q \times A \times Sen(\Sigma) \times 2^{\mathcal{SE}(\mathcal{L})} \times Q$*.*

A small specification example is given in [2]. Transition systems are specifications of reactive systems. Given a signature morphism $\sigma : (\Sigma, A) \rightarrow (\Sigma', A')$ and a specification $\mathcal{S}' = (Q', \mathbb{T}')$ over (Σ', A'), $Spec(\sigma)(\mathcal{S}')$ is the specification $\mathcal{S} = (Q, \mathbb{T})$ over (Σ, A) such that $Q = Q'$ and $\mathbb{T} = \{(q, a, \varphi, \delta, q') | (q, \sigma(a), Sen(\sigma)(\varphi), \sigma(\delta), q') \in \mathbb{T}'\}$.

5.1.2 Semantics

Semantics of transition systems are defined by Kripke frames themselves defined as follows:

Definition. 20 (Kripke frame) *Given a signature* $\mathscr{L} = (\Sigma, V, A)$*, an* Kripke frame *over* $\mathscr{L}$ *or* $\mathscr{L}$*-model, is a couple* $(\mathcal{W}, R)$ *where:*

- $\mathcal{W}$ *is a* I*-indexed set* $(\mathcal{W}^i)_{i \in I}$ *of* Σ*-models such that* $W^i_s = W^j_s$ *for every* $i, j \in I$ *and* $s \in S$*, and*
- R *is a* A*-indexed set of "accessibility" relations* $R_a \subseteq I \times I$*.*

Here, states are defined by Σ-models. Therefore, side-effects will consist on moving from a Σ-model to another one by changing the semantics of functions according the assignments given in the set δ of transitions. Formally, this is defined as follow: if $\mathcal{A}$ is a Σ-model, then $\_^{\mathcal{A}} : T_F \rightarrow \mathcal{A}$ is the Σ-morphism inductively defined by $f(t_1, \ldots, t_n) \mapsto f^{\mathcal{A}}(t_1^{\mathcal{A}}, \ldots, t_n^{\mathcal{A}})$

Definition. 21 (Side-effect semantics) *Let* $\mathscr{L} = (\Sigma, V, A)$ *be a signature. Let* $\mathcal{A}$ *and* $\mathcal{B}$ *be two* Σ*-models. We note* $\mathcal{A} \rightsquigarrow_\delta \mathcal{B}$ *to mean that the state* $\mathcal{A}$ *is transformed into the state* $\mathcal{B}$ *along* δ*, if and only if* $\mathcal{B}$ *is defined as* $\mathcal{A}$ *except that for every* $t \mapsto t' \in \delta$ *(resp.* $p(t_1, \ldots, t_n) \mapsto b$*),* $t^{\mathcal{B}} = t'^{\mathcal{A}}$ *(resp.* $(t_1^{\mathcal{A}}, \ldots, t_n^{\mathcal{A}}) \in p^{\mathcal{B}}$ *iff* $b = true$*).*

Definition. 22 (Semantics of transition systems) *Given a transition system* $\mathcal{S} = (Q, \mathbb{T})$ *over a signature* $\mathscr{L}$*, the semantics for* $\mathcal{S}$*, noted* $Real(\mathcal{S})$*, is the set of all the Kripke frames* $(\mathcal{W}, R)$ *over* $\mathscr{L}$ *such that the set of indexes* $I = Q$*, and satisfying both implications:*

1. $(q, a, \varphi, \delta, q') \in \mathbb{T} \wedge \mathcal{W}^q \models \varphi \wedge \mathcal{W}^q \rightsquigarrow_\delta \mathcal{W}^{q'} \Rightarrow q \; R_a \; q'$
2. $q \; R_a \; q' \Rightarrow \exists (q, a, \varphi, \delta, q') \in \mathbb{T}, \mathcal{W}^q \models \varphi \wedge \mathcal{W}^q \rightsquigarrow_\delta \mathcal{W}^{q'}$

Hence, the way whose dynamic is dealt with in this paper follows the state-as-algebra style [21, 3] where states are Σ-models and state transformations are transitions from a state-model to another state-model.

5.1.3 Synchronous product

Synchronous product combines two transition systems into a single one by synchronizing transitions. Understandably, executions of synchronous product modelize system behavior as a synchronizing concurrent system. Hence, when an action a is "executed" in the product, then every component with a in its alphabet must execute a transition labeled with a. Formally, the synchronous product of two transition systems is defined as follows:

Definition. 23 (Synchronous product) *Let* $\mathcal{S}_i = (Q_i, \mathbb{T}_i)$ *be a transition system over a signature* $\mathscr{L}_i = (\Sigma_i, V_i, A_i)$ *with* $i = 1, 2$ *such that:*

- *for every transition* $(q_1, a, \varphi_1, \delta_1, q_1') \in \mathbb{T}_1$ *and every* $f(t_1, \ldots, t_n) \mapsto t_1' \in \delta_1$ *(resp.* $p(t_1, \ldots, t_n) \mapsto b \in \delta_1$*), there does not exist a transition* $(q_2, a, \varphi_2, \delta_2, q_2') \in \mathbb{T}_2$ *and a side-effect* $t_2 \mapsto t_2' \in \delta_2$ *with* t_2 *of the form* $f(t_1', \ldots, t_n')$ *(resp.* $p(t_1', \ldots, t_n') \mapsto b' \in \delta_2$*),*
- *and conversely, that is this condition on side-effects has also to be satisfied by replacing* $\mathbb{T}_1$ *by* $\mathbb{T}_2$*,* δ_1 *by* δ_2 *and* δ_2 *by* δ_1*.*

The synchronous product *of* $\mathcal{S}_1$ *and* $\mathcal{S}_2$*, noted* $\mathcal{S}_1 \otimes \mathcal{S}_2$*, is the transition system* $(Q, \mathbb{T})$ *over* $\mathscr{L} = (\Sigma_1 \cup \Sigma_2, V_1 \cup V_2, A_1 \cup A_2)$ *defined as follows:*

- $Q = Q_1 \times Q_2$
- *if* $a \in A_1 \cap A_2$*,* $(q_1, a, \varphi_1, \delta_1, q_1') \in \mathbb{T}_1$ *and* $(q_2, a, \varphi_2, \delta_2, q_2') \in \mathbb{T}_2$ *then* $((q_1, q_2), a, \varphi_1 \wedge \varphi_2, \delta_1 \cup \delta_2, (q_1', q_2')) \in \mathbb{T}$
- *if* $a \in A_1 \setminus A_2$ *and* $(q_1, a, \varphi_1, \delta_1, q_1') \in \mathbb{T}_1$ *then for every* $q_2 \in Q_2$*,* $((q_1, q_2), a, \varphi_1, \delta_1, (q_1', q_2)) \in \mathbb{T}$

- *if* $a \in A_2 \setminus A_1$ *and* $(q_2, a, \varphi_2, \delta_2, q_2') \in \mathbb{T}_2$ *then for every* $q_1 \in Q_1$, $((q_1, q_2), a, \varphi_2, \delta_2, (q_1, q_2')) \in \mathbb{T}$

Both conditions on side-effects allow us to remove the case where for an identical function name f (resp. a predicate p) applied to an identical tuple of arguments yields different values, and then causes the functionality of f (resp. makes inconsistent the set of side-effects resting on p) to fail.

By following the notions of our abstract framework, the synchronous product gives rise to the connector $Sync$. To define this connector, we consider the shape I composed of three nodes i, j and k and two arrows $a_1 : i \to j$ and $a_2 : i \to k$. The connector $Sync$ is then defined for every diagram δ where $\delta(i)$ is the empty transition system over the signature $(\Sigma_\emptyset, A_i)$ where $\Sigma_\emptyset$ is the empty **FOL**-signature, $\delta(j) = (Q_j, \mathbb{T}_j)$ over the signature (Σ_j, A_j) and $\delta(k) = (Q_k, \mathbb{T}_k)$ over the signature (Σ_k, A_k), and yields $Sync(\delta) = \delta(j) \otimes \delta(k)$ over the signature (Σ, A) with the co-cone $p : Sig \circ \delta \to (\Sigma, A)$ which is the pushout of $Sig(\delta(a_1))$ and $Sig(\delta(a_2))$ in Sig.

5.2 Results

The synchronous product of two transition systems $\mathcal{S}_1 \otimes \mathcal{S}_2$ have generally true emergent properties. The reason is the set $Mod(Th(\mathcal{S}_1^\bullet \cup \mathcal{S}_2^\bullet))$ of Kripke frames may be greater than $Real(\mathcal{S}_1 \otimes \mathcal{S}_2)$. Indeed, Kripke frames in $Real(\mathcal{S}_1 \otimes \mathcal{S}_2)$ have to preserve the shape of the transition system $\mathcal{S}_1 \otimes \mathcal{S}_2$ unlike Kripke frames in $Mod(Th(\mathcal{S}_1^\bullet \cup \mathcal{S}_2^\bullet))$. Hence, properties in $(\mathcal{S}_1 \otimes \mathcal{S}_2)^\bullet$ may be more numerous than in $Th(\mathcal{S}_1^\bullet \cup \mathcal{S}_2^\bullet)$. However, we can show under some conditions that non-conformity properties cannot occur along synchronous product. More precisely, we are going to show that the "only if" part of the conformity property is satisfied but the "if" part only holds when formulas that label transitions are conditional equations (i.e. expressed in the logic **CEL**).

Let us start by showing that the semantic consequences of $\mathcal{S}_1$ and $\mathcal{S}_2$ are preserved by $\mathcal{S}_1 \otimes \mathcal{S}_2$. Let us suppose a $\mathcal{S}_1 \otimes \mathcal{S}_2$-model $(\mathcal{W}, R)$, and let us define a $\mathcal{L}_i$-model $(\mathcal{W}_i, R_i)$ for $i = 1, 2$ as follow:

- for every $q \in Q_i$, $\mathcal{W}_i^q = Mod(\Sigma_i \hookrightarrow \Sigma)(\mathcal{W}^{(q,q')})$ for any $q' \in Q_j$ with $j \neq i \in \{1, 2\}$
- $\forall a \in A_i$, $R_{i_a} = \{(q, q') | \exists \varphi \in Sen(\Sigma_i), \exists \delta \in \mathcal{SE}(\mathcal{L}),\ (q, a, \varphi, \delta, q') \in \mathbb{T}_i\}$

Let us note Γ_i for $i = 1, 2$, the set of all these $\mathcal{L}_i$-models.

Theorem. 3 *Each* $(\mathcal{W}_i, R_i) \in \Gamma_i$ *is a* $\mathcal{S}_i$*-model.*

Proof. *The first condition of Definition 22 is obvious. To prove the second condition, let us suppose a transition* $(q, a, \varphi, q') \in \mathbb{T}_i$*. By construction, there exists a transition* $((q, q_j), a, \varphi', \delta', (q', q_j')) \in \mathbb{T}$ *such that either* $\varphi' = \varphi$ *and* $\delta' = \delta$*, or* $\varphi' = \varphi \wedge \varphi''$ *and* $\delta' = \delta \cup \delta''$*. In both cases, by hypothesis, we have that* $(\mathcal{W}^{(q,q_j)} \models \varphi'$*. Therefore, by the satisfaction condition for* **FOL** $\mathcal{W}_i^q \models \varphi$*. Moreover, by the condition on side-effects in Definition 23, we have that* $\mathcal{W}_i^q \rightsquigarrow_\delta \mathcal{W}_i^{q'}$

Proposition. 2 $\forall \iota : V \to W$,
$(\forall (\mathcal{W}_i, R_i) \in \Gamma_i,\ \forall q \in Q_i, (\mathcal{W}_i, R_i) \models_\iota^q \varphi)$
$\implies (\forall q_j \in Q_j,\ (\mathcal{W}, R) \models_\iota^{(q,q_j)} \varphi)$

Proof. *By induction on the structure of* φ*.*

Basic case. φ *is of the form* $p(t_1, \ldots, t_n)$*. Let* $q_j \in Q_j$*. By definition, there exists* $(\mathcal{W}_i, R_i) \in \Gamma_i$ *such that* $\mathcal{W}_i^q = Mod(\Sigma_i \hookrightarrow \Sigma)(\mathcal{W}^{(q,q_j)})$*. By hypothesis, we have* $\mathcal{W}_i^q \models_\iota p(t_1, \ldots, t_n)$*, and then* $\mathcal{W}^{(q,q_j)} \models_\iota p(t_1, \ldots, t_n)$*.*

General case. *Let us handle the case where* φ *is* $\Box_a \varphi'$*. Let us suppose that* $(\mathcal{W}, R) \models_\iota^{(q,q_j)} \varphi$*. Then, let us consider* (q', q_j) *such that* $(q, q_j)\ R_a\ (q', q_j')$*. By the hypothesis, we have for every* $(\mathcal{W}_i, R_i) \in \Gamma_i$ *that* $(\mathcal{W}_i, R_i) \models_\iota^q \varphi$*. By construction, we also have* $q\ R_{i_a}\ q'$ *for every* $(\mathcal{W}_i, R_i) \in \Gamma_i$*. Therefore, for every* $(\mathcal{W}_i, R_i) \in \Gamma_i$*,* $(\mathcal{W}_i, R_i) \models_\iota^q \varphi'$*, and then by the induction hypothesis, we have* $(\mathcal{W}, R) \models_\iota^{(q',q_j')} \varphi'$*, whence we can conclude* $(\mathcal{W}, R) \models_\iota^{(q',q_j')} \varphi$*.*

The cases of Boolean connectives and quantifier are simpler and left to the interested reader.

Theorem. 4 $\mathcal{S}_i^\bullet \subseteq (\mathcal{S}_1 \otimes \mathcal{S}_2)^\bullet$

Proof. *Let* $\varphi \in \mathcal{S}_i^\bullet$*, and let* $(\mathcal{W}, R) \in Real(\mathcal{S}_1 \otimes \mathcal{S}_2)$*. Let* $\iota : V \to W$ *be an interpretation. By Theorem 3, for every model* $(\mathcal{W}_i, R_i) \in \Gamma_i$*, we have* $(\mathcal{W}_i, R_i) \models \varphi$*, and then for every* $q \in Q_i$ *we also have* $(\mathcal{W}_i, R_i) \models_\iota^q \varphi$*. Therefore, by Proposition 2, we have for every* $q_j \in Q_j$ *that* $(\mathcal{W}, R) \models_\iota^{(q,q_j)} \varphi$*, and then* $(\mathcal{W}, R) \models \varphi$*.*

To show the "if" part of the conformity property, we need to make some restrictions on formulas that label transitions. Hence, we suppose that transition systems are built over the logic **CEL**, and then given a model $(\mathcal{W}, R)$ of transition system $\mathcal{S}$, for each $q \in Q$, $\mathcal{W}^q$ is now an algebra. Therefore, the logic for transition systems is the modal first-order logic defined as in Section 2 except that now Σ-atoms are restricted to Σ-equations.

Given two transition systems $\mathcal{S}_1$ and $\mathcal{S}_2$ over the signatures $\mathcal{L}_1$ and $\mathcal{L}_2$, respectively, and satisfying the above restriction, for $i \neq j \in \{1, 2\}$, and for every $(\mathcal{W}_j, R_j) \in Mod(\mathcal{S}_j)$ we define the mapping $\mathcal{F}_{(\mathcal{W}_j, R_j)} : Mod(\mathcal{S}_i) \to Mod(\mathcal{L})$ where $\mathcal{L}$ is the signature over which the transition system $\mathcal{S}_1 \otimes \mathcal{S}_2$ is built as follow: if we note for a Σ-algebra $\mathcal{A}$, $th(\mathcal{A}) = \{\varphi | \varphi : \textbf{CEL}\text{-formula}, \mathcal{A} \models \varphi\}$, then to every

$(\mathcal{W}_i, R_i)$, $\mathcal{F}_{(\mathcal{W}_j,R_j)}((\mathcal{W}_i, R_i)) = (\mathcal{W}, R)$ such that $(\mathcal{W}, R)$ is the $\mathcal{L}$-model defined by [9]

- $\forall q \in Q_i,\ \forall q' \in Q_j,\ W^{(q,q')} = T_{\Gamma_i/\Gamma}(\mathcal{W}_i^q) \times T_{\Gamma_j/\Gamma}(\mathcal{W}_i^q)$
- $R_a = \{((q_1, q_1'), (q_2, q_2'))|\exists\varphi \in Sen(\Sigma), \exists\delta \in \mathcal{SE}(\mathcal{L}),\ ((q_1, q_1'), a, \varphi, (q_2, q_2')) \in \mathbb{T}\}$

where $\Gamma_i = th(\mathcal{W}_i^q)$, $\Gamma_j = th(\mathcal{W}_j^{q'})$, and $\Gamma = th(\mathcal{W}_i^q) \cup th(\mathcal{W}_j^{q'})$.

Theorem. 5 *For every $(\mathcal{W}_j, R_j) \in Mod(\mathcal{S}_j)$ and every $(\mathcal{W}_i, R_i) \in Mod(\mathcal{S}_i)$, $\mathcal{F}_{(\mathcal{W}_j,R_j)}((\mathcal{W}_i, R_i))$ is a $\mathcal{S}_1 \otimes \mathcal{S}_2$-model.*

Proof. *The first condition of Definition 20 is obvious. To prove the second condition, let us suppose a transition $((q_1, q_1'), a, \varphi, \delta, (q_2, q_2')) \in \mathbb{T}$. By construction, φ and δ are:*

1. *either of the form $\varphi' \wedge \varphi''$ with $\varphi' \in Sen(\Sigma_i)$ and $\varphi'' \in Sen(\Sigma_j)$ and $\delta' \cup \delta''$ with $\delta' \in \mathcal{SE}(\mathcal{L}_i)$ and $\delta'' \in \mathcal{SE}(\mathcal{L}_j)$,*
2. *or $\varphi \in Sen(\Sigma_i) \cup Sen(\Sigma_j)$ and $\delta \in \mathcal{SE}(\mathcal{L}_i) \cup \mathcal{SE}(\mathcal{L}_j)$.*

This then leads to the two following cases:

1. *Suppose that φ is of the form $\varphi' \wedge \varphi''$ and then $\delta = \delta' \cup \delta''$. This means by construction, that $(q_1, a, \varphi', \delta', q_1') \in \mathbb{T}_i$ and $(q_2, a, \varphi'', \delta'', q_2') \in \mathbb{T}_j$. By hypothesis, we have $\mathcal{W}_i^{q_1} \models \varphi'$ and $\mathcal{W}_j^{q_1'} \models \varphi''$. Therefore, we have that $T_{\Gamma_i/\Gamma}(\mathcal{W}_i^{q_1}) \models \varphi' \wedge \varphi''$ and $T_{\Gamma_j/\Gamma}(\mathcal{W}_j^{q_2}) \models \varphi' \wedge \varphi''$, and then so is $T_{\Gamma_i/\Gamma}(\mathcal{W}_i^{q_1}) \times T_{\Gamma_j/\Gamma}(\mathcal{W}_j^{q_2})$ (recall that conditional equations are preserved along the cartesian product of algebras). Moreover, by hypothesis, we also have that $\mathcal{W}_i^{q_1} \rightsquigarrow_{\delta'} \mathcal{W}_i^{q_2}$ and $\mathcal{W}_j^{q_2} \rightsquigarrow_{\delta''} \mathcal{W}_j^{q_2'}$. By definition, Γ_i (resp. Γ_j) contains the ground equational theory of $\mathcal{W}_i^{q_1}$ (resp. $\mathcal{W}_j^{q_2}$). If we note $\Gamma_i' = th(\mathcal{W}_i^{q_1'})$, $\Gamma_j' = th(\mathcal{W}_j^{q_2'})$ and $\Gamma' = th(\mathcal{W}_i^{q_1'}) \cup th(\mathcal{W}_j^{q_2'})$, then we have $T_{\Gamma_i/\Gamma}(\mathcal{W}_i^{q_1}) \rightsquigarrow_\delta T_{\Gamma_i'/\Gamma'}(\mathcal{W}_i^{q_2})$ and $T_{\Gamma_j/\Gamma}(\mathcal{W}_j^{q_2}) \rightsquigarrow_\delta T_{\Gamma_j'/\Gamma'}(\mathcal{W}_j^{q_2'})$.*
2. *The case where $\varphi \in Sen(\Sigma_i) \cup Sen(\Sigma_j)$ and $\delta' \in \mathcal{SE}(\mathcal{L}_i)$ and $\delta'' \in \mathcal{SE}(\mathcal{L}_j)$ is noticeably similar to the previous one.*

MFOL is closed under isomorphism. Moreover, by Theorem 3, $Real$ is compatible with each morphisms p_i of the co-cone p associated to the connector $Sync$. Finally, by Proposition 2 and Theorem 4, RSC is satisfied. Therefore, Theorem 6 is a specialization of Theorem 2.

Theorem. 6 *If for every $(\mathcal{W}_i, R_i) \in Mod(\mathcal{S}_i)$, every $(\mathcal{W}_j, R_j) \in Mod(\mathcal{S}_j)$, and every $q \in Q_i$ and every $q' \in Q_j$, the adjunct morphism $\mu_{\mathcal{W}_i^q} : \mathcal{W}_i^q \to Mod(\Sigma_i \hookrightarrow \Sigma)(T_{\Gamma_i \setminus \Gamma}(\mathcal{W}_i^q))$ is an isomorphism, then $(\mathcal{S}_1 \otimes \mathcal{S}_2)^\bullet \cap Sen(\mathcal{L}_i) \subseteq \mathcal{S}_i^\bullet$.*

Proof. *Let $\varphi \in (\mathcal{S}_1 \otimes \mathcal{S}_2)^\bullet \cap Sen(\mathcal{S}_i)$ and let $(\mathcal{W}_i, R_i) \in Mod(\mathcal{S}_i)$. By Theorem 5, for every $(\mathcal{W}_j, R_j) \in Mod(\mathcal{S}_j)$, we have that $\mathcal{F}_{(\mathcal{W}_j,R_j)}((\mathcal{W}_i, R_i)) \models \varphi$. As the adjunct morphism $\mu_{\mathcal{W}_i^q}$ is an isomorphism, for every $\iota : V \to W_i$ there exists $\iota' : V \to T_{\Gamma_i/\Gamma}(W_i) \times T_{\Gamma_j/\Gamma}(W_j)$ such that $\iota = p_i \circ \iota'$ where p_i is the i-th projection map $p_i : T_{\Gamma_i/\Gamma}(\mathcal{W}_i^q) \to T_{\Gamma_i/\Gamma}(\mathcal{W}_i^q) \otimes T_{\Gamma_j/\Gamma}(\mathcal{W}_j^{q'})$ for $q \in Q_i$ and $q' \in Q_j$. By hypothesis, for every $q \in Q_i$ and every $q' \in Q_j$, $\mathcal{F}_{(\mathcal{W}_j,R_j)}((\mathcal{W}_i, R_i)) \models_{\iota'}^{(q,q')} \varphi$. It is then easy to show by induction on the structure of φ that $(\mathcal{W}_i, R_i) \models_\iota^q \varphi$.*

Example. 4 *When dealing with formulas expressed in the logic* **CEL** *to label transitions, we often make restrictions on algebras denoting states. Indeed, to allow inductive proofs or for computability reasons, state-algebras are then restricted to reachable [10] or some quotients of the ground term algebra. Let us the suppose for the below counter-example of the conditions given in Theorem 6, that we restrict our approach to state-algebras defined by reachable algebras. Let us consider the two following transition systems $\mathcal{S}_1$ and $\mathcal{S}_2$ defined respectively over the two following signatures $\mathscr{L}_1$ and $\mathscr{L}_2$:*

[9] **Cartesian product and preservation results** Let Σ be a signature, I be a set and $(\mathcal{A}_i)_{i\in I}$ be a I-indexed family of Σ-algebras. Let us note $\prod_{i\in I} \mathcal{A}_i$ the Σ-algebra defined as follow:

- for every $s \in S$, its carrier of sort s is $\prod_{i\in I} (A_i)_s$,
- for every $f : s_1 \times \ldots \times s_n \to s \in F$, $f^{\prod_{i\in I} \mathcal{A}_i}$ is the mapping that to every $(a_1, \ldots, a_n) \in \prod_{i\in I} (A_i)_{s_1} \times \ldots \times \prod_{i\in I} (A_i)_{s_n}$, associates $(f^{\mathcal{A}_i}(a_1^i, \ldots, a_n^i)|i \in I)$ where given $a \in \prod_{i\in I} (A_i)_s$, a^i is the ith coordinate of a.

By construction, we can notice that:

$$\prod_{i\in I} \mathcal{A}_i \models_\iota \varphi \iff \forall i \in I,\ \mathcal{A}_i \models_{\iota^i} \varphi$$

where for every interpretation ι, ι^i is the interpretation defined by $x \mapsto a^i$ if $\iota(x) = a$. It is well-known that conditional equations are preserved by Cartesian product of algebras, that is, if for every $i \in I$, $\mathcal{A}_i \models \Gamma \Rightarrow \alpha$, then $\prod_{i\in I} \mathcal{A}_i \models \Gamma \Rightarrow \alpha$.

[10] A Σ-algebra is *reachable* if, and only if the unique Σ-morphism $\mu : T_F \to \mathcal{A}$ is surjective, that is all the values in A are denoted by the evaluation of a ground term.

$$\Sigma_1 = \left\{ \begin{array}{l} S = \{nat\}, \\ F = \left\{ \begin{array}{l} 0 :\rightarrow nat; \\ s : nat \rightarrow nat, \\ + : nat \times nat \rightarrow nat \end{array} \right\}, \\ P = \emptyset \end{array} \right.$$

$$\Sigma_2 = \left\{ \begin{array}{l} S = \{nat\}, \\ F = \{0 :\rightarrow nat; s, p : nat \rightarrow nat\}, \\ P = \emptyset \end{array} \right.$$

$A_1 = A_2 = \{a\}$

Let us define $\mathcal{S}_1$ and $\mathcal{S}_2$ as follows:

- $\mathcal{S}_1 = (\{q_1, q_2\}, \{q_1 \stackrel{a,\varphi_1,\delta_1}{\longrightarrow} q_2\})$ *where* $\varphi_1 = (s(x) = s(y) \Rightarrow x = y) \wedge x + 0 = x \wedge x + s(y) = s(x + y)$ *and* $\delta_1 = \emptyset$. $\mathcal{S}_2 = (\{q_1', q_2'\}, \{q_1' \stackrel{a,\varphi_2,\delta_2}{\longrightarrow} q_2'\})$ *where* $\varphi_2 = (s(x) = s(y) \Rightarrow x = y) \wedge s(p(x)) = x \wedge p(s(x)) = x$ *and* $\delta_2 = \emptyset$.

By definition of $\mathcal{S}_1$ (resp. $\mathcal{S}_2$), the unique $\mathcal{S}_1$-model (resp. $\mathcal{S}_2$-model) is $(\mathcal{W}, R)$ where $\mathcal{W}_1^{q_1} = \mathcal{W}_1^{q_2} = \mathbb{N}$ (resp. $\mathcal{W}_2^{q_1'} = \mathcal{W}_2^{q_2'} = \mathbb{Z}$). On the contrary, by construction, in $\mathcal{S}_1 \otimes \mathcal{S}_2$, we have the transition $(q_1, q_1') \stackrel{a,\varphi',\delta'}{\longrightarrow} (q_2, q_2')$ where $\varphi' = \varphi_1 \wedge \varphi_2$ and $\delta' = \emptyset$, and then all the $\mathcal{S}_1 \otimes \mathcal{S}_2$-model satisfy $\mathcal{W}^{(q_1,q_1')} = \mathcal{W}^{(q_2,q_2')} = \mathbb{Z}$. Consequently, the modal formula $\varphi' \Rightarrow \Box_a(\forall x.\exists y.x + y = 0)$ belongs to $(\mathcal{S}_1 \otimes \mathcal{S}_2)^\bullet$ but not in $\mathcal{S}_1^\bullet$. The reason is $F_{(\mathcal{W}_2,R_2)}(\mathcal{W}_1^{q_1}) = \mathbb{Z}$. Therefore, the adjunct functor $\mu_{\mathcal{W}_1^{q_1}}$ is injective but not surjective, and then is not an isomorphism.

6 Conclusion

In this paper, our main contribution is twofold. First, we have formally defined the notion of emergent properties independently of formalism, and of the form of both specifications and architectural connectors. Secondly, we have studied in this abstract framework, some general conditions that enable us to obtain two general properties that guarantee when a system is not complex. These conditions are based on the category theory of morphism conservativeness and adjunction. Finally, to illustrate our abstract framework, we have instantiated our abstract framework with reactive component-based systems described by transition systems and combined together through synchronous product, and we have applied our general results to obtain global systems lacking of non-conformity properties which have been recognized as being the cause of bad interactions between components.

An ongoing research that we are currently pursuing is to extend abstract connectors to heterogeneous abstract connectors, that is connectors defined on component specifications described in heterogeneous formalisms. For this purpose, we will take benefit from [11, 23] and from works that we made on hierarchical heterogeneous specifications [9].

References

[1] M. Aiguier, P. Le Gall, and M. Mabrouki. A formal definition of complex software. In *ICSEA 2008: Proceedings of the 2008 The Third International Conference on Software Engineering Advances*, pages 415–420. IEEE Computer Society, 2008.

[2] M. Aiguier, P. Le Gall, and M. Mabrouki. Emergent properties in reactive systems. In *APSEC 2008: Proceedings of the 2008 15th Asia-Pacific Software Engineering Conference*, pages 273–280. IEEE Computer Society, 2008.

[3] M. Aiguier. Étoile-specifications: An object-oriented algebraic formalism with refinement. *Journal of Logic and Computation*, 14(2):145–178, 2004.

[4] M. Aiguier, C. Gaston, and P. Le Gall. Feature logics and refinement. In *APSEC 2002: Proceedings of the 9th Asian Pacific Software Engeenering Conference*, pages 385–395. IEEE Computer Society Press, 2002.

[5] R. Allen. *A Formal Approach to Software Architecture*. PhD thesis, Carnegie Mellon, School of Computer Science, Junuary 1997. Issued as CMU Technical Report CMU-CS-97-144.

[6] R. Allen and D. Garlan. A formal basis for architectural connectors. *ACM TOSEM*, 6(3):213–249, 1997.

[7] L. Blass, P. Clements, and R. Kasman. *Software Architecture in Practice*. Addison Wesley, 1998.

[8] T. Borzyszkowski. Logical systems for structured specifications. *Theoretical Computer Science*, 286:197–245, 2002.

[9] S. Coudert and P. Le Gall. A reuse-oriented framework for hierarchical specifications. In *AMAST 2000: Proceedings of the 8th International Conference on Algebraic Methodology and Software Technology*, pages 438–453, London, UK, 2000. Springer-Verlag.

[10] R.-I. Damper. Emergence and levels of abstraction. *International Journal of Systems Science*, 31(7):811–818, 2000. Editorial for the Special Issue on 'Emergent Properties of Complex Systems'.

[11] R. Diaconescu. Grothendieck institutions. *Applied Categorical Structures*, 10(4):383–402, 2002.

[12] R. Diaconescu. Jewels of institution-independent model theory. In K. Futatsugi, J.-P. Jouannaud, and J. Meseguer, editors, *Algebra, Meaning, and Computation, Essays Dedicated to J.-A. Goguen on the Occasion of His 65th Birthday*, volume 4060 of *Lecture Notes in Computer Science*. Springer-Verlag, 2006.

[13] A.-C. Ehresmann and J.-P. Vanbremeersch. *Memory Evolutive Systems: Hierarchy, Emergence, Cognition*. Elsevier Science, 2007.

[14] H. Ehrig, M. Balmadus, and F. Orejas. New concepts for amalgation and extension in the framework of specification logics. In *AMAST 1991: Algebraic Methodology and Software Technology*, Lecture Notes in Computer Science. Springer, 1991.

[15] J.-L. Fiadeiro. *Categories for Software Engineering*. Springer-Verlag, 2004.

[16] J.-L. Fiadeiro, A. Lopes, and M. Wermelinger. A mathematical semantics for architectural connectors. In R.-C. Backhouse and J. Gibbons, editors, *Generic Programming*, volume 2793 of *Lecture Notes in Computer Science*, pages 178–221. Springer-Verlag, 2003.

[17] D. Garlan, R.-T. Monroe, and D. Wile. Acme: An architecture description interchange language. In *CASCON 1997: Proceedings of the 1997 conference of the Centre for Advanced Studies on Collaborative research*, pages 169–183. IBM Press, 1997.

[18] C. Gaston, M. Aiguier, and P. Le Gall. *Language Constructs for Decsribing Features*, chapter Algebraic treatment of feature-oriented systems, pages 105–125. Springer-Verlag, 2000.

[19] J. Goguen. *Advances in Cybernetics and Systems Research*, chapter Categorical Foundations for General Systems Theory, pages 121–130. Transcripta Books, 1973.

[20] J. Goguen and R.-M. Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the ACM*, 39(1):95–146, 1992.

[21] Y. Gurevich. Evolving algebras 1993: Lipari guide. In *Specification and Validation Methods*, pages 9–36. Oxford University Press, 1995.

[22] J.-V. Guttag and J.-J. Horning. The algebraic specification of abstract data types. *Acta Informatica*, pages 27–52, 1978.

[23] T. Mossakowski. Institutional 2-cells and grothendieck institutions. In *Essays Dedicated to Joseph A. Goguen*, volume 4060 of *Lecture Notes in Computer Science*, pages 124–149. Springer, 2006.

[24] F. Orejas. *Algebraic Foundations of Systems Specification*, chapter Structuring and Modularity, pages 159–201. IFIP State-of-the-Art Reports. Springer, 1999.

[25] D. Perry and A. Wolf. Foundations for the study of software architectures. *ACM SIGSOFT Software Engineering Notes*, 17(4):40–52, 1992.

[26] M. Plath and M. Ryan. Feature integration using a feature construct. *Science of Computer Programming*, 41(1):53–84, 2001.

[27] A. Sernadas, C. Sernadas, and C. Caleiro. Denotational semantics of object specification. *Acta Informatica*, 35(9):729–773, 1998.

[28] A. Tarlecki. Moving between logical systems. In M. Haveraaen, O. Owe, and O.-J. Dahl, editors, *Recent Trends in Data Type Specifications. 11th Workshop on Specification of Abstract Data Types*, volume 1130 of *Lecture Notes in Computer Science*, pages 478–502. Springer Verlag, 1996.

[29] A. Tarlecki. *Algebraic Foundations of Systems Specification*, chapter Institutions: An abstract Framework for Formal Specifications, pages 105–131. IFIP State-of-the-Art Reports. Springer, 1999.

A Underlay System for Enhancing Dynamicity within Web Mashups

Heiko Pfeffer
Technische Universität Berlin
Franklinstr. 28/29, 10587 Berlin, Germany
heiko.pfeffer@tu-berlin.de

Abstract

Rich Internet Applications (RIA) and composed Web applications, referred to as Mashups, have become the new generation of Web based applications, aggregating multimedia data such as audio, video and images from multiple providers and combining them to more powerful and value-added applications. In the same time, mobile devices such as smartphones or PDAs are increasingly used to access the Web and Web based applications. Therefore, Web developers nowadays face a huge client heterogeneity, where devices differ in their computing capabilities, screen size, available bandwidth, and mobility pattern.

This paper introduces a service composition model that can underlie modern Web applications and can be executed within a respective runtime at every browser-enabled client. This model allows software develops to create applications by abstracting from concrete services and APIs, which are incorporated dynamically during runtime. Thus, the resulting Underlay System for Web Mashups overcomes device heterogeneity by providing means to execute the application logic in a unified runtime while integrating the concrete service implementations based on the current context of the user and the respective client device.

Keywords: *Service Composition, Mashups, Composed Web Applications, Workflows, Timed Automata, Realtime*

1 Introduction

In 2003, Macromedia coined the term of Rich Internet Applications (RIA), describing a new breed of upcoming Web applications that combines the best of desktop software, the Web, and advanced communication technologies [11]. Here, the aim is to build highly interactive Web applications that feature a broad spectrum of data such as audio, video, images and text in order to enhance the experience of the user. Lawton later introduced an update on the definition of RIA, highlighting the importance of the usage of Web technologies [18]. At the same time, composed Web applications, i.e. applications that were built by the incorporation of content from multiple 3rd party service providers, referred to as Mashups, gained importance. On ProgrammableWeb.com [1], 1318 open APIs and 3985 Mashups were registered in May 2009. The most prominent motives for offering data to communities of users and software developers are the generation of new ideas by independent developers, the search for new revenue streams and the opportunity to draw traffic to the providers' sites and remain or become a dominant player in the Web. Through publicly available APIs the community often develops new features without assistance from the providers, posing the possibility of vertical integration for the providers.

However, the development of such Web Mashups as a composition of 3rd party services is aggravated by differing interface descriptions of 3rd party services and their varying return types that considerably reduce the amount of reusable code and forestall an easy swapping of services with similar functionality. Thus, today's composed Web applications remain tailor-made for specific purposes.

Other domains have already faced similar challenges and responded with respective mechanisms. For instance, architecting distributed applications in open systems with SOA-based software patterns has proven as good practice to achieve robustness and extensibility. Inter-component dependencies are loosened to on-demand selection and purpose-driven interaction at runtime, service consuming and service providing entities stay autonomic; this central concept is often referred to as late binding of services.

However, SOA based applications incorporate a huge protocol stack and are thus a bad fit for modern Web applications based on the REST architectural style [13]. Moreover, SOAs are tailor-made to support large-scale business processes where end-users are not the center of attention as opposed to Web Mashups. Business processes completely lack a presentation layer to display results of computations and to interact with the user. Moreover, they do not provide suitable means to remain responsive to user preferences or context or to support a broad device heterogeneity,

i.e. mobile users accessing Web applications through small portable devices with restricted computing capabilities.

Within the remainder of this work, a Underlay System for Web Mashups is introduced that considerably reduces the problems entailed by a high device heterogeneity by incorporating services based on the current context during runtime. Here, section 2 first gives a general introduction to composed applications, highlighting results from the world of SOA and the Web, respectively. The conceptual part of section 3 discusses the Underlay System itself and builds upon results given in [29], while section 4 highlights an architecture for realizing the proposed approach.

2 State of the Art and Related Work

This section gives a short introduction to the most important concepts relevant within the scope of this work and relates it to similar approaches. While subsection 2.1 focuses on the concept of service composition and workflow languages, subsection 2.2 deals with a respective grounding to the Web architecture, discussing Mashups as a poster child for composed Web applications.

2.1 Service Composition

Service Oriented Architecture (SOA) and Web Services have successfully addressed the problem of Service Composition for business processes by introducing appropriate composition languages [5], enabling the controlled execution of multiple accumulated Web Services. However, those technologies such as UDDI and SOAP strictly rely on fix infrastructures for service discovery and provisioning. First steps towards perpetuating the success of SOA and Web Services to mobile networks have been achieved by providing service frameworks for accessing Web Services from mobile devices such as PDAs and Smartphones [23]. A static service infrastructure for service discovery and provisioning is nevertheless still indispensable. Successful Service Composition methods remain tailor-made for business processes [17].

Within the world of SOA and Web Services, various models for service composition are present. The currently most prominent one is BPEL4WS [3, 8], a combination of IBM's graph-based Web Service Flow Language (WSFL) [19] and Microsoft's block-oriented XLANG [30]. Beside, many other composition languages exist, wherefrom none has made the breakthrough to an holistically accepted standard for Web Service Composition [33]. In order to provide more dynamic service composition approaches featuring a functional-based service discovery and binding, some approaches have replaced the Web Service Description Language (WSDL) [6] by semantic service descriptions such as OWL-S [31]. Especially for mobile devices, more lightweight descriptions have been proposed in order to reduce the computation complexity entailed by semantic reasoning processes [27].

However, dynamic service composition faces the difficulty that new semantic service descriptions have to be generated that can be used for future service discovery and binding. The approach introduced in this paper therefore aims at extending the flexibility of service compositions achieved trough late binding mechanisms to the structure of service composition plans. This proceeding entails the generation of multiple equivalent or similar service compositions with regard to their functionality, which may however differ in their non-functional properties.

2.2 Mashups - Composed Web Applications

Mashups are composed Web applications integrating data from multiple 3rd party providers to provide a value-added functionality and experience to the user. Within this section, we first outline the three basic roles within Mashup based Web applications and then discuss two different Mashups styles that denote possible ways to realize them, building upon [21] and [22] mainly.

2.2.1 Mashup Roles

In general, Mashups architectures feature three elementary roles as illustrated in Figure 1. The Content Provider is a source of data that can be accessed through open APIs over various Web protocols such as REST, RSS or SOAP. The Mashup Site is the new build Web application that requests content and services from various data sources and combines them in order to provide a value-added application to the user. The Client is the interface to the user (presented within a Web-browser). The user can interact with the Mashup through client-side scripting languages such as JavaScript.

Within the following section, we will first focus on the role of content providers, introducing famous and successful services that are exposed through open APIs. Here, special emphasis will be put on the way they expose their APIs, the types of data they return and how their integration can characterize the resulting Mashup. Thereafter, we will discuss basic styles of Mashups, highlighting the different responsibilities of clients and Mashup sites within the respective approaches.

2.2.2 Mashup Styles

In general, it is distinguished between two basic styles for creating and executing Mashups, entailing different responsibilities of the client and the Mashup site in order to execute the actual Mashup. Both styles expose serious advan-

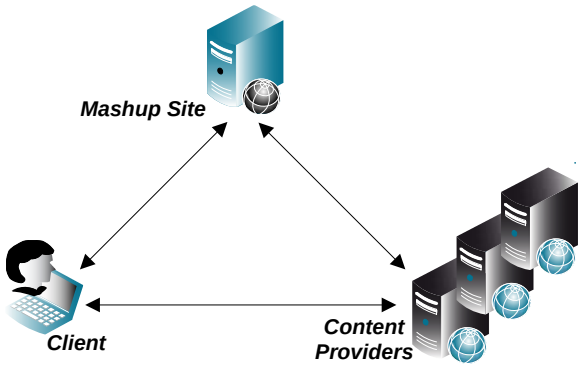


Figure 1. General Roles of a Mashup Web Application.

tages and disadvantages with regard to their performance, load distribution and security, and thus have to be pondered against each other carefully with regard to the requirements of the respective Mashup application.

Server-side Mashups Within server-side Mashups, services and content are integrated at a server, which plays the role of a proxy between the Web application on the client and other Web sites that are integrated into the Mashup. Every request or event originating from the client is forwarded to this proxy server, which then makes the calls on the respective Web sites. Because of this central server role acting as a proxy, server-side Mashups are often referred to as Proxy Mashups. Figure 2 shows the general setting of a server-side Mashup.

Whenever a user generates an event at the client Web browser, e.g. pushes a button or clicks on a map, the event triggers a JavaScript function (#1). This Java Script function makes a HTTP request to the Mashup site (#2). Commonly, this request is an Ajax request; it will later be discussed in greater detail. The request is received by a Web component such as a Servlet or Java Server Page (JSP) on the Mashup site. Based on the received request, the component calls one or multiple methods within a class or multiple classes containing the application logic to make calls on 3rd party APIs (#3). Because of their role of acting as a proxy between the client's request and the request to the actual Mashup servers, these classes are often referred to as Proxy Classes. Note that proxy classes are not restricted in the way they are realized, thus, they can be implemented as plain Java classes or large-scale J2EE components. Thus, the Proxy classes connect to the addressed Mashup servers and request the respective services (#4). In turn, the Mashup servers process the request and respond with the resulting data (#5). The proxy classes receive the response and can process the data before forwarding it to the Web component (#6). This option of processing data on the server side enables most of the advantages of a server-side Mashups. For instance, data can be transformed in a format such as JSON that is easier to process by the client, it can integrate different data sources and only send the integrated piece of data to the client, or data can be buffered or cached in order to increase the performance of succeeding requests. Finally, the Web component sends the response to the client (#7). Here, the view of the page at the client side is updated according to the response. In case the initial request has been an Ajax request in form of an XMLHttpRequest object, this page update is achieved by the callback function within the XMLHttpRequest that manipulates the Document Object Model (DOM) accordingly.

Client-side Mashups Within client-side Mashups, the integration of the single services and data sources is performed at the client instead of using an additional proxy server. Thus, clients connect directly to 3rd party APIs in order to request services; a client-side Mashup is abstractly depicted in Figure 3.

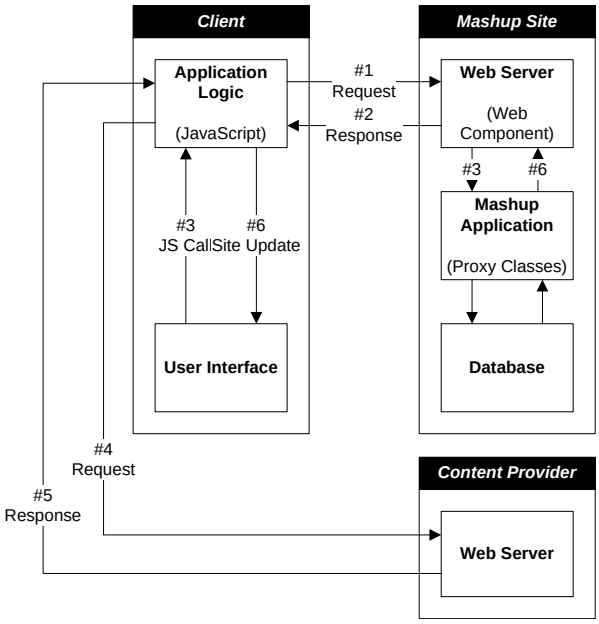


Figure 3. Client-side Mashup.

Initially, the client's browser makes a request to the Mashup site (#1), initiating the server to load the requested Web page into the client (#2). This Web page provides access to JavaScript libraries that enable the client later to directly call services at 3rd party APIs without addressing the Mashup site beforehand. There are three common ways to provide access to JavaScript libraries. First, the Web

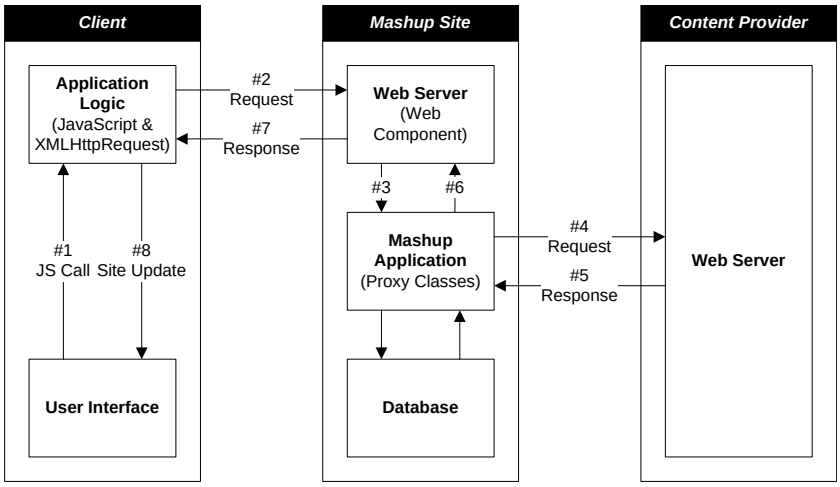


Figure 2. Server-side Mashup.

page may reference the JavaScript library from the respective 3rd party service provider such as Google Maps; here, it is sufficient to reference the library by a valid URL. In case the 3rd party provider does not expose an appropriate JavaScript library, the Mashup developer can provide one on his own and make it available on the Mashup site. Alternatively, other 3rd party libraries may be referenced to ease the Mashup's creation. Specific actions on the Web page trigger the browser to call a function in the JavaScript library integrated within the Web page. This function dynamically creates a <script> tag in the Web site that points to the according 3rd party server (#3). Afterwards, the client initiates an HTTP request based on the ¡script¿ tag including the desired format of the response (#4). As discussed above, the format of the response provided by a service can commonly vary, ranging from XML to YAML or JSON. For client-side Mashups, JSONP (JSON with Padding) is the most common response format since it can be easily evaluated (by the JavaScript function *eval()*) at the client. JSONP extends JSON by the capability of appending the name of a local callback function to the JSON object. Thus, in case the server provides the response in JSONP, a call is made on the callback function with the JSON object as parameter (#6). Finally, the DOM of the page is manipulated by the JavaScript function and the page is updated accordingly.

3 Underlay System for Web Mashups

Within this section, a Underlay System for Web Mashups is specified by introducing a service composition model that can describe and execute composed Web applications, i.e. Mashups, and is -in the same time- responsive to the environmental user context. Special emphasis is put on overcoming the high degree of device heterogeneity we face in today's Web.

3.1 Distinguishing Mashups and Business Processes

Business processes and Web Mashups both constitute composed applications. However, they are created with different requirements. While business processes are tailor-made to describe large-scale business transactions among multiple enterprises, Mashups are rapidly created Web applications for end-users aiming at a high degree of interactivity and graphical presentation, where content from 3rd party providers is combined to add value to the created Web application.

The fact that Mashups are built on top of the Web protocol stack while business processes are designed for SOAs is also reflected in the underlying technologies for remote service invocation and service interworking as shown in Table 1.

Moreover, Table 1 outlines the focus of both technologies. Business processes are defined by service composition languages such as BPEL and provide late binding mechanisms for services that are described by languages such as WSDL and stored in repositories where they can be accessed via UDDI. Thereby, enterprises can modify the implementations of their single services (e.g. upgrade them to a new version) without any need to notify service consumers of their changes since the services are incorporated by their descriptions only and thereby decoupled from the actual service implementation. Mashups on the contrary do not provide any kind of controlled service execution or late binding of services. Instead, Web developers directly integrate the concrete APIs of 3rd party providers into their

Feature	*Business Processes (SOA)*	*Web (Mashups)*
Remote Invocation	WebServices (WSA), XML-RPC	XMLHttpRequest (Ajax), JSON RPC, COMET
Service Interworking	.NET, J2EE	EcmaScript (Java Script)
Service Composition	**Yes**	**No**
Composition Languages	BPEL4WS, WSFL, WSCDL	
Composition Execution Engines	Bexee, Oracle BPEL Process Manager, IBM WBISF	
Late Binding	**Yes**	**No**
Service Description Languages	WSDL, WSDL-S, OWL-S	
Service Discovery	UDDI	
Interaction	**No**	**Yes**
Presentation		HTML, CSS, XML, Streamed audio and video
User Input		HTML Forms, XForms, JS Keyboard/Mouse Event Models

Table 1. Business Processes vs. Mashups

Mashups. In return, Mashups provide a connection to the user by default: Web applications are build by a Web page that consists of HTML, CSS and Java Script and therefore possess a presentation and means to interact with the user by nature. Application logic is only invoked when an event is created by the user that triggers a certain functionality.

The following can be concluded: While business processes enable software developers to create the application logic of a process without dealing with its graphical presentation and possible user interaction, Mashups are created as interactive graphic presentations that react on user input and thereby invoke small parts of application logic that modifies the presentation of the application.

3.2 Requirements for a Mashup Underlay System

The objective of this work is to incorporate the ability of SOA business processes to structure the underlying application logic as a service composition and to dynamically incorporate services or APIs through late binding mechanisms in order to overcome the device heterogeneity of today' Web Mashups. Moreover, this techniques allow for a more dynamic adaptation of Mashups to the users' needs and their environmental context by providing means to include services based on the users' preferences, on the client device and its respective capabilities, and the current performance of the single services.

Given these requirements, the following components are identified as central for the Underlay System for Web Mashups:

- **Workflow** - A workflow graph enables software developers to specify the execution order of the single services and APIs.

- **Dataflow** - A dataflow graph allows developers to connect data retrieved as output of one service to the inputs of other services.

- **Abstract Service Access** - An abstraction layer for services and APIs defines a unified way to access services providing the same resource, i.e. the same object on the presentation and interaction layer. Thus, this layer allows developers to conceptually include a map within their Mashups without specifying the concrete service that is creating this presentation. This procedure does not only free the software developer from programming against multiple APIs, but also allows for a dynamic exchange of the bound services or APIs in case the context of the user changes.

- **Context Awareness** - The Underlay System provides an interface to develop selection mechanisms in case multiple services are available for the same presentation. For instance, in case both Google Maps and Yahoo Maps are identified to be capable of presenting a map to the user, the selection can be made dependent on user preferences or context.

- **Time Awareness** - To overcome device heterogeneity, time is identified as critical context information that can trigger the evaluation of the appropriateness of integrated services. For instance, in case the processing of a service exceeds a predefined time limit, it can be concluded that the processing power of the device is too low and that the service should be replaced. The same technique can be applied to identify services that

are causing too much traffic given the available bandwidth. For instance, one could define that in case the response time of a service exceeds a certain time span, the traffic has to be too high for the currently available bandwidth such that the service has to be replaced with a less demanding one.

Within the next subsections, a service composition model is developed on a theoretical level that builds the core of the Underlay System for Web Mashups and can be executed within a respective runtime both on servers and browser-enabled clients. More insights on the related realization of the Underlay System is given later in section 4.

3.3 Modeling Service Compositions

This section deals with the representation and execution control of service compositions. Therefore, subsection 3.3.1 defines a service model assumed for the software components serving as basis for service compositions provides a general overview of the two main representation paradigms chosen for service compositions. The graph-based representation of the workflow and the dataflow within those compositions is then discussed in subsections 3.3.2 and 3.3.3, respectively. Finally, the interworking of both graphs is exemplarily outlined in subsection 3.3.4.

3.3.1 Service Model and Basic Concepts

Services themselves are considered as atomically executable parts of application logic, whereby the execution of the service is independent from outer computations and data structures. This assumption is in-line with the REST architectural style of the Web, where services are defined as stateless. We rely on IOPE descriptions characterizing a service functionality by its inputs, outputs, preconditions and effects as discussed by Jaeger et al. [16]. Inputs and outputs of a service are distinguishable by a unique identifier referred to as a *port*. Since it is aim at introducing a representation of service composition plans that is independent from the service descriptions they rely on, possible semantically enhanced description of preconditions or effects are not specified. Instead, we assume effect to be present or not, thus, we identify every effect with a boolean indicating whether the effect has already been generated or not. Preconditions can then be pondered against those boolean representations of service effects.

To represent service compositions properly as well as to control their execution, a bipartite graph concept based on a workflow graph controlling the execution of the single components within a service composition and a dataflow graph defining the passage of data between services' output and input ports is introduced. Each transition of the workflow graph corresponds to a service containing I/O parameters and a set of effects it generates during execution as described above.

In case a services precondition is met, the service can be executed, thus, an action representing the service functionality is performed, which consumes the service's input and generates a finite set of effects and a finite set of outputs. Transitions between locations are optionally annotated with guards, which restrict the passage of the transition by requiring the satisfaction of specific preconditions or previously generated outputs. Notably, the output of a service is not necessarily required as input for a service reached by the next transition within the workflow graph; instead, an output may also become relevant after multiple other services have been executed. Therefore, a second graph is defined, specifying the dataflow within a service composition. This dataflow graph shares the set of locations with the workflow graph, but represents the flow of outputs from one service component to the input of another with its transitions. A guard for passing a transition within a workflow graph that would lead to the execution of the next service can thus depend on the presence of a set of effects (expressed through preconditions) and on the availability of all inputs that have to be created as outputs of other services before. Notably, the dataflow graph is not necessarily completely connected, thus may be a set of graphs.

In the following section, the workflow graph of a service composition is formally defined, specifying its guard based transition behavior and its time awareness.

3.3.2 Workflow Graph

The workflow graph is supposed to control the execution order of single services within a service composition. Based on the requirements identified in subsection 3.2, automata theory is proposed as the mathematical basis for service composition representation. In [24], a transformation from UML state machines [10] to timed automata has already proven that automaton theory can be easily accessed by user-friendly modeling languages such as UML, enabling the feasibility of the presented approach. Coevally, automaton theory already provides rapid modifications of the automaton structure by simple operations, because automata are represented as basic digraphs. In order to provide realtime consideration within service compositions, the definition of workflow graphs borrows from timed automata. Timed automata [2, 9] are finite automata [4, 12] extended by a set of real-time valued clocks. In the following, we will refer to the definition provided by Clarke et al. in [7]. Let X be a finite set of real-valued variables standing for clocks. Clock constraints are then defined as follows.

Definition 3.1 (Clock Constraints) *A clock constraint is a conjunctive formula of atomic constraints of the form* $x * n$

or $x - y * n$ for $x, y \in X$, $* \in \{\leq, <, =, >, \geq\}$ and $n \in \mathbb{N}$. $\mathcal{C}(X)$ denotes the set of clock constraints. If φ_1 is in $\mathcal{C}(X)$, then $\varphi_1 \wedge \varphi_2$ is also in $\mathcal{C}(X)$.

A Timed Automaton is then defined as follows.

Definition 3.2 (Timed Automaton) *A timed automaton $\mathcal{A}$ is a 6-tuple $\{\Sigma, S, s_0, X, I, T\}$, where*

1. *Σ is a finite alphabet (standing for actions),*
2. *S is a finite set of locations,*
3. *$s_0 \in S$ are the initial locations (also called starting locations),*
4. *X is a set of clocks,*
5. *$I : S \rightarrow \mathcal{C}(X)$ assigns invariants to locations, i.e., provides a mapping from locations to clock constraints, and*
6. *$T \subseteq S \times \mathcal{C}(X) \times \Sigma \times 2^X \times S$ is the set of transitions.*

Abbreviatory, we will write $s \overset{g,a,\lambda}{\rightarrow} s'$ for $\langle s, g, \alpha, \lambda, s' \rangle$, i.e. the transition leading from location s to s'. The transition is restricted by the constraint g (often called guard); $\lambda \subseteq X$ denotes the set of clocks that is reset during the transition passage.

Notably, many model checker such as UPPAAL [32], operating on timed automata restrict the location invariants to downwards closed ones, i.e. do only allow invariants of the form $x \leq n$ or $x < n$ for $n \in \mathbb{N}$.

Infinite state transition systems (sometimes called infinite state transition graphs) are used as model for a timed automaton $\mathcal{A}$. Deep introductions into the notion of transition systems can be found in [15, 20]. Here, the definition of Clarke et al. [7] will be followed again, specifying an infinite state transition system $\mathcal{T}(\mathcal{A})$ for a given timed automaton $\mathcal{A}$ as a 4-tuple $\mathcal{T}(\mathcal{A}) = \{\Sigma, Q, Q_0, R\}$. A state $q \in Q$ of this transition system is defined as a pair (s, v), where $s \in S$ is a location and $v : x \rightarrow \mathbb{R}^+$ is a clock assignment. The initial states are identified by all initial locations, where all clocks are set to zero, i.e. $Q_0 = \{(s, v) | s \in S_0 \wedge \forall x \in X\, [v(x) = 0]\}$.

The definition of the state transition relation is implied by the following two needs. First, a notion is required to reset a clock to zero, i.e., for $\lambda \in X$, we define $v[\lambda := 0]$ for mapping all clocks in λ to zero. For $d \in \mathbb{R}$, we define $v + d$ as a clock assignment that maps the current value of v to $v(x) + d$ for all clocks $x \in X$. Based on this, two different types of transitions are defined, covering the passage of time and the triggering of actions. Time can pass while the system is in a specific location as long as the according state invariant is not violated. This elapsing of time is referred to as *delayed transition*, specified as $(s, v) \overset{d}{\rightarrow} (s, v+d), d \in \mathbb{R}^+$, subjected to the condition that the invariant $I(s)$ is not violated for every $v + e, 0 \leq e \leq d$. The second type of transitions describes the actual execution of an action and is thus referred to as *action transition*. If there is a transition $\langle s, \alpha, g, \lambda, s' \rangle$ where v satisfies g and $v' = v[\lambda := 0]$, we note $(s, v) \overset{\alpha}{\rightarrow} (s', v')$ for a transition with $\alpha \in \Sigma$. Together, we receive the transition relation $\mathcal{R}$ for $\mathcal{T}(\mathcal{A})$ as $(s, v)\mathcal{R}(s', v')$ (also written as $(s, v) \overset{\alpha}{\Rightarrow} (s', v')$), if there are s'' and v'' so that $(s, v) \overset{d}{\rightarrow} (s'', v'') \overset{\alpha}{\rightarrow} (s', v')$ for some $d \in \mathbb{R}$. Thus, an action α can also be performed without elapsing time, i.e. in case $d = 0$. Moreover α may also stand for the empty action $\tau \in \Sigma$, i.e. a transition can also be passed without entailing the execution of an specific action.

Since actions are supposed to model the behaviour of a service, the execution of an action α is mapped to the consumption of inputs, the creation of outputs and the generations of effects. Inputs and outputs are regarded as typed variables bounded to specific ports, enabling the definition of I/O passage by means of a dataflow graph. The domains of variables thus constitute their primitive data type.

The Workflow Graphs Θ are now modeled as timed automata as defined in Definition 3.2, extended by the ability of actions $\alpha \in \Sigma$ to operate on typed variables and effects. Typed variables are defined as 2-tuples $(x, \Gamma(x))$, where x is a variable and $\Gamma(x)$ its respective domain. For now, only variables x with $x \in R \subset \mathbb{R}$ are considered. Let $\mathcal{V}$ be a finite set of typed variables. Regarding the handling of effects, we define $\mathcal{E}$ as a final set of variables $\gamma^* \in \{0, 1\}$ indicating whether an effect γ has been created (γ^* set to 1) or not (γ^* set to 0). We assume that $\mathcal{V}$ contains at least all variables of $\mathcal{E}$, thus, $\mathcal{E} \subseteq \mathcal{V}$.

Definition 3.3 (Guard Constraints) *A real-value constraint is a propositional logic formula $x * n$, where $x \in \mathcal{V}$ is a typed variable, $n \in \mathbb{R}$ and $* \in \{<, \leq, \geq, >, ==\}$. $\mathcal{C}(R \backslash \mathcal{V})$ denotes the set of real-value constraints containing only variables of $\mathcal{V}$.*

If φ_1 and φ_2 are in $\mathcal{C}(R \backslash \mathcal{V})$, $\varphi_1 \wedge \varphi_2$, $\varphi_1 \vee \varphi_2$ and $\neg\varphi_1$ are also in $\mathcal{C}(R \backslash \mathcal{V})$.

Let $\varphi_c \in \mathcal{C}(X)$ be a clock constraint as defined in Definition 3.1, φ_r a real-value constraint.

A guard constraint is a formula $\varphi = \varphi_c | \varphi_r | \varphi_c \wedge \varphi_r$ evaluating to a Boolean. $\mathcal{C}(X \circ R \backslash \mathcal{V})$ denotes the set of all guard constraints.

The workflow graph is now defined as a timed automaton whose actions and guards can also operate on typed variables.

Definition 3.4 (Workflow Graph) *A workflow graph Θ is a timed automaton as defined in Definition 3.2, where*

1. Σ *is a finite alphabet. The alphabet represents the actions which are identified by the single services within the service composition.*

2. S *is a finite set of locations defining the service compositions's current state,*

3. $s_0 \in S$ *are the initial locations defining the initial state of the service composition,*

4. X *is a set of clocks,*

5. $I : S \rightarrow \mathcal{C}(X)$ *assigns invariants to locations, restricting the system in the amount of time it is allowed to remain in the current state, and*

6. $T \subseteq S \times \mathcal{C}(X \circ R \backslash \mathcal{V}) \times \Sigma \times 2^X \times S$ *is the set of transitions, denoting the execution of a service (represented by an action α). The passage of a transition (and thus the execution of a service) thereby depends on whether the according guard is met.*

In order to decide whether an output of a service component is required as an input for another one, a dataflow graph is introduced in the following section.

3.3.3 Dataflow Graph

Dataflow graphs represent the connections of output and input ports. They keep the same locations as the workflow graph introduced in Definition 3.4 and use labeled transitions to express inter-component data passing.

Definition 3.5 (Dataflow Graph Ω) *A dataflow graph Ω is a labeled directed digraph $\Omega = \{N, P, E\}$, where,*

1. N *is a final set of labeled nodes, which is equivalent to the set of locations S held in the according workflow graph Θ,*

2. P *is a set of port mappings represented by 2-tuples $p = (p_1, p_2)$ indicating the passage of the output from port p_1 to the input port p_2, and*

3. $E \subseteq N \times P \times N$ *is a final set of labeled directed transitions.*

Each location represents the data generated by the execution of action α_i; we therefore label a node with $d(\alpha_i)$ indicating the output data from action α_i. If a transition is passed within a workflow graph Θ entailing the execution of action α_i, all outgoing transitions from location $d(\alpha_i)$ within the according dataflow graph Ω are passed. A transition passage $d(\alpha_i) \overset{(p_m,p_n)}{\rightarrow} d(\alpha_j)$ within Ω effectuates that the output at port p_m from action α_i is redirected to input port p_n of action α_j in case action α_i is executed within Θ.

A service composition is specified by its workflow and dataflow graph, i.e., is defined as a 2-tuple $\langle \Theta, \Omega \rangle$.

The following section discusses a small example for service composition representation and control.

3.3.4 Workflow and Dataflow Graph Interworking

The service composition to be represented is abstractly depicted in Figure 4.

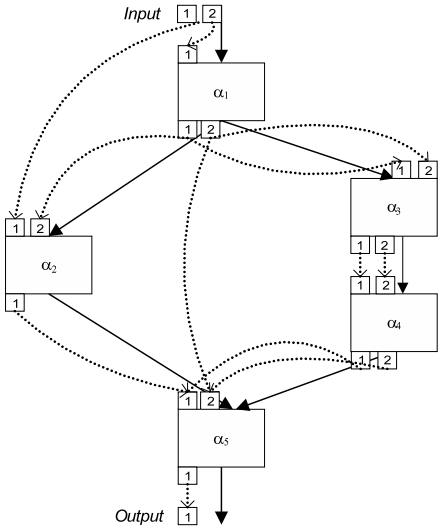


Figure 4. A Possible Service Composition.

The involved service components are represented by rectangles labeled with an action α_i describing their functionality. The order of execution, i.e., the workflow, is depicted by bold arrows while the passage of inputs and outputs is indicated by dashed arrows. They connect the output ports of service components with input port of other components that are drawn as numbered squares.

The example service composition contains five service components (labeled with $\alpha_1, ..., \alpha_5$).

After α_1 has been started, either α_2 or α_3 and α_4 are executed. In case both execution paths are enabled, i.e. all premises in terms of I/O and effect availability are met, a path is chosen nondeterministically according to the notion of transition systems.

While service executions in common service composition representations such as in Figure 4 are represented as nodes within graphs, the one introduced in this paper models service executions as actions performed during transitions between locations. The workflow graph deduced from the example service composition is illustrated in Figure 5 (a).

Guards restricting the transition from a location s to a location s' operate on the generated effects (through pre-

conditions) and a final set of inputs and outputs. Outputs are not stored but redirected to input ports by means of the according dataflow graph. The dataflow graph depicted in Figure 5 (b) is supposed to belong to the workflow graph illustrated in Figure 5 (a). For instance, the execution of

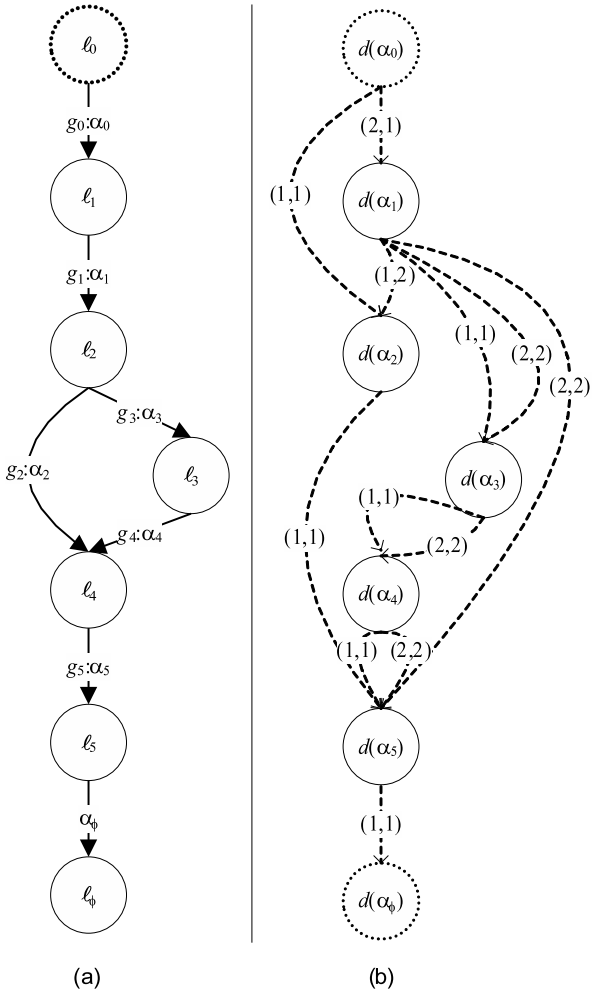


Figure 5. (a) An according workflow graph. (b) A dataflow graph representing the I/O Passing.

α_1 implies the redirection of the output from the first output port of α_1 to the second input port of α_2 and the first input port of α_3. Moreover, the output from the second output port of α_1 is forwarded to the second input port of α_3 and to the second input port of α_5.

3.3.5 Enabling Parallel Execution

Because of their relation to automata and their given transition relation introduced in section 3.3.2, workflow graphs already support a wide set of control structures required to guide the execution order of services; the three most important ones are abstractly depicted in Figure 6. First,

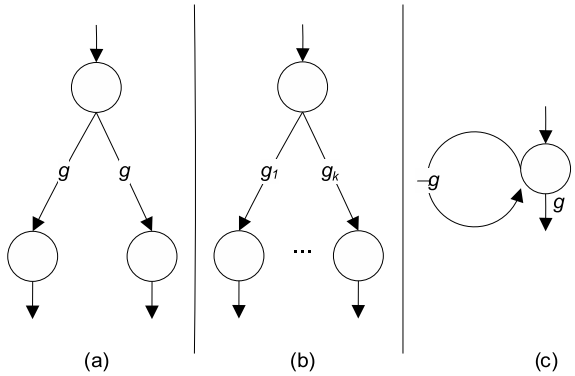


Figure 6. Basic control structures of automata semantics. (a) Nondeterministic Selection. (b) Choice. (c) Looping.

nondeterministic decisions between multiple possible service invocations are modeled with a finite set of outgoing transitions labeled with the same guard g. In case the guard is met, all transitions are enabled and the transition relation discussed in section 3.3.2 selects a transition nondeterministically. Decisions such as *if-else* or *switch* statements are represented by multiple outgoing transitions labeled with different guards $g_1, ..., g_k$. Here, deadlocks can occur in case all guards are evaluated as *false* under a given set of constraints. Moreover, multiple guards may be *true*, entailing a non-deterministic decision making between the enabled transitions. Last, looping can be modeled with transitions pointing back to the origin state as depicted in Figure 6 (c).

However, workflow graphs do not provide all means to model parallel execution of services. By passing a transition, the according service is executed. Note that the introduced model does not assume that the workflow graph remains in the location reached by the transition passage until the service's execution is finished. Instead, the next transition is directly passed (in case at least one transition's guards are met), entailing the execution of the next service. Thereby, service execution becomes parallel. If both services have been started on the same device, the local scheduling algorithm ensures their pseudo-parallel execution. In case they are executed on different nodes and therefore multiple processors, they are running in parallel. However, this approach fails in case the biggest subset of a given set of services should be executed in parallel.

Assume two services α_1 and α_2 are selected for parallel execution. In case the services are initiated in the order

$\alpha_1 \rightarrow \alpha_2$, it may be possible that α_1 cannot be started since an input is missing. The blocking of the transition would entail that also service α_2 is not started, although its guard may be met. Therefore, a control structure for parallel execution of services is required. The left-hand side of Figure 7 depicts an abstract workflow reflecting the previously described situation of two services α_1 and α_2 that should be executed in parallel.

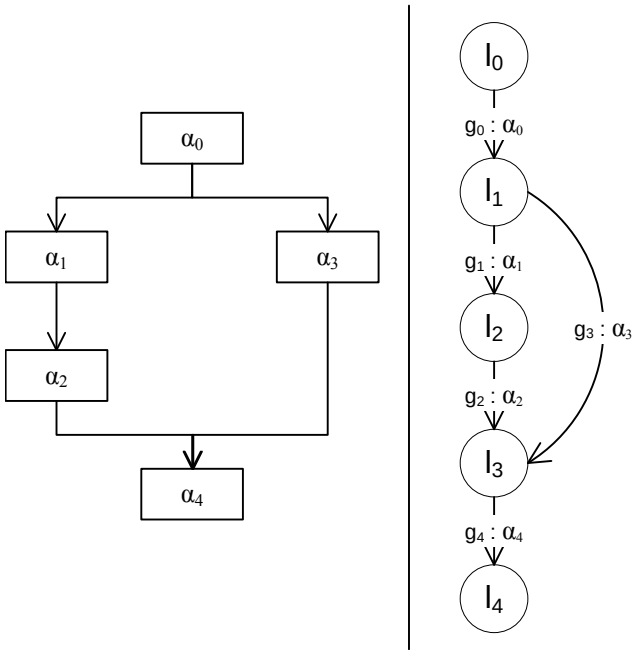


Figure 7. Timed automaton structure ensuring parallel execution.

The right-hand side of the same Figure depicts the timed automaton that corresponds to the abstract service composition. Here, the execution of α_2 would be forestalled in case guard g_1 is not met although g_2 may be fulfilled. In order to unblock this behavior, an additional idle state is automatically inserted during the generation of the workflow graph that is entered after every execution of a service that may be executed in parallel to other services. The accordingly modified workflow graph is illustrated in Figure 8.

Because of this modification, the timed automaton moves into the idle state as soon as the guard of the next service is not met, such that the invocation of another service can be initiated whose guards is fulfilled. The utilization of semaphores ensures that every service is only executed once.

Thus, by inserting a special control structure that can be generated for a given finite set of services, workflow graphs possess the same expressiveness as other workflow languages such as BPEL, while relying on sound formal

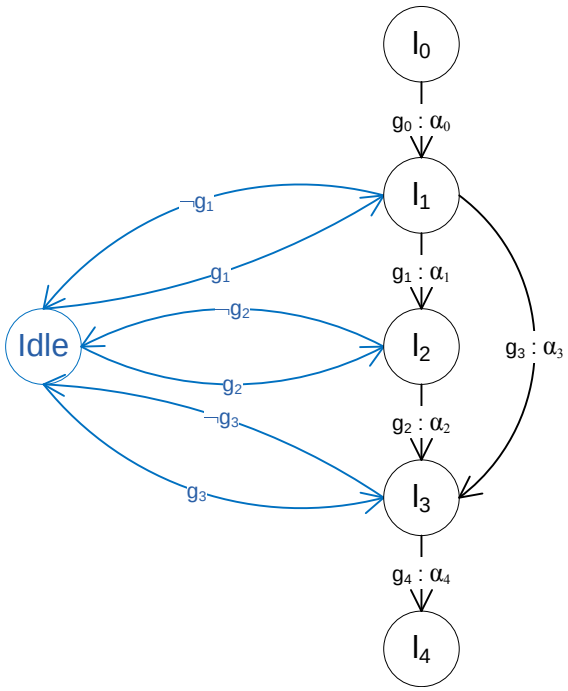


Figure 8. Timed automaton structure ensuring parallel execution.

model featuring realtime consideration and flexibility.

4 An Architecture featuring a Underlay System for Web Mashups

Within this section, the previously introduced formal method for the representation of workflow-based service compositions within a Underlay System for Web Mashups is realized as an architecture enabling the utilization of workflows for the representation of Web applications and features the late binding of 3rd party APIs. This architecture is implemented according to the REST architectural style, the architectural style of the Web, that was proposed by Fielding within his PhD thesis and is known for its related realization within the Web, HTTP [13].

Figure 9 gives an high-level overview of the proposed architecture.

Here, the Mashup site, i.e. the Mashup proxy, is extended by 4 key components. First, a workflow engine for the previously introduced model based on timed automata enables the structured execution of service compositions. In addition to version for servers written in Java, the runtime is also implemented in pure Java Script that can be transferred to any browser enabled client during initialization such that the workflows can be executed on the client side and in-

corporate local services residing on the client. A detailed description of the implementation was presented in [25]. The single services of the service composition are given as abstract triples *<res, act, attr[]>*. Here, *res* describes the resource that should result from the respective service execution, *act* defined an action that should be performed on the resource *res*, and *attr[]* denotes a set of attributes that are required for the execution of the specified action. Thus, the triple *<map, center, lat=23432, long=92834>*

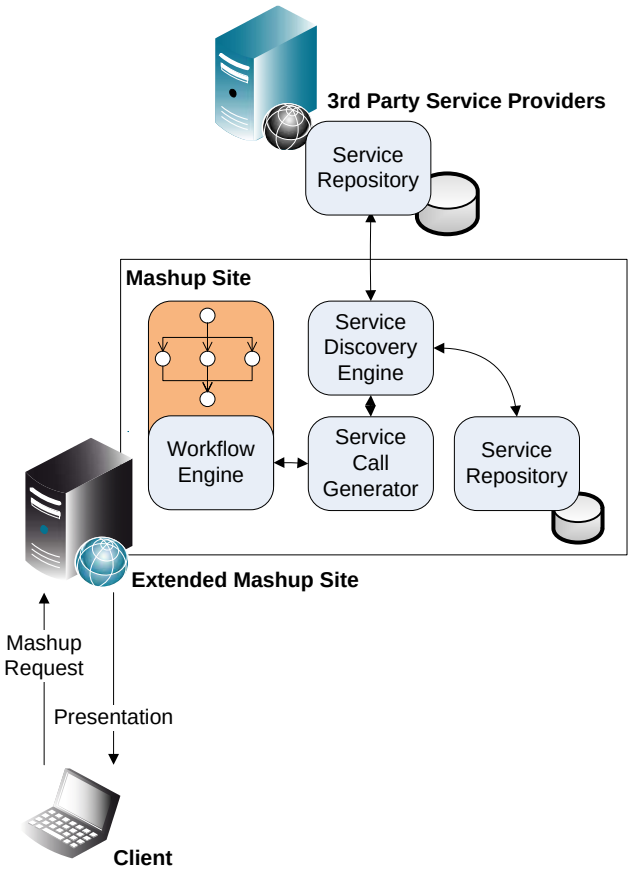


Figure 9. Server-side Service Composition Engine for Composed Web Applications: Architecture.

denotes that a map is requested that is centered on the location given by the longitude *long* and latitude *lat*. In general, a location is not known by its longitude and latitude, but by its name. Therefore, another service may be requested by the triple *<location, getCoordinates, location=Berlin>* that generates the longitude and latitude values for the city "Berlin". The output of this service can thus be used as input for the next service. Of course, the attribute "Berlin" can also be given during runtime by a user input.

This proceeding interprets the workflow model in a REST conform way: The developer requests a resource and is returned a representation of the resource that depends on the given action and set of attributes. For instance, while the triple *<location, getCoordinates, location=Berlin>* returns the location of a given city by its longitude and latitude coordinates, the triple *<location, getStreetName, location=Peter's Pub>* may return a String contain the street name of a bar called "Peter's Pub".

To enable such an abstraction from actual service implementations, the proposed architecture features a late binding engine. Here, 3rd party provides can describe their exposed APIs featuring REST interfaces with the Web Application Description Language (WADL) [14]. (The integration of other services such as classic SOAP Web Services described by WSDL [6] are also supported but are not described in greater detail here.) The late binding engine is capable of matching the previously introduced triples denoting abstract descriptions of a requested resource to these WADL descriptions and return a set of WADL files that describe services which can generate the requested resource. For instance, a triple *<location, getCoordinates, location=Berlin>* may be matched by Google Maps, Yahoo Maps, and suchen.de. One of these WADL based descriptions can now be passed to the Service Call Generator where a REST call for the chosen service is automatically created and handed back to the workflow engine where the service can be invoked.

This proceeding enables more responsiveness and dynamicity of Web Mashups on multiple levels. First, service interfaces are encapsulated behind abstract descriptions which reduce the amount of coding for the software developer. Thus, instead of dealing with the concrete API of Google Maps, software developers can interface with the abstract notion of triples, which is conform to the REST architectural style of the Web. Thereby, the integration of multiple APIs providing a similar functionality can be avoided. For example, a developer that has always used Google Maps to integrate a map into his or her Web application may decide to use Yahoo Maps instead. In this case, the developer would have to incorporate a complete new API into his or her Mashup, although both services provide the same functionality.

However, the dynamic binding of services does not only reduce the development time for Mashups, but also enables the consideration of device capabilities, user profiles, context information, and Quality of Service (QoS) parameters of services. Thus, in case the late binding engine returns multiple WADL files, i.e., has discovered multiple services or APIs that can provide the requested resource, the service may be explicitly selected that matches the current circumstances best. For instance, in case a user has defined in his profile that he prefers services from Google because of the

familiar user interface, Google Maps may be bound instead of Yahoo Maps. On the other hand, a monitoring entity may have noticed that the response time of Yahoo has been considerably shorter than from Google, leading to the incorporation of Yahoo Maps instead of Google Maps; this incorporation of non-functional properties if services has been introduced in [26]. It also enables the recovery of services that expose a weak performance during runtime, i.e. in case a service is not responding or responding too slow, it may be dynamically replaced during runtime by another service.

In addition to the flexibility provided by the dynamic integration of single service implementations within the service composition, the workflow graph itself may be adapted in its structure to as a response to environmental changes; this adaption has already been introduced in [28].

Thus, the architecture proposed in this work provides means to overcome the problem of heterogeneous clients by providing means to bind and replace services dynamically before and during runtime based on non-functional properties of services, context information, device capabilities or user profiles.

5 Conclusion and future Prospect

Within this paper, a Underlay System for Web Mashups was introduced that combined the advantages of SOA business processes and the rich presentation and interaction capabilities of today's Mashups to overcome the challenge of device heterogeneity and context dependence in modern Web applications. Therefore, a formal model for service compositions based on a bipartite graph concept has been introduced that consists of a workflow graph defined as a timed automaton over an extended finite set of typed variables and a dataflow graph specifying the passage of data between the single services. By introducing a special structure of a timed automaton, the semantics of classic automata were extended to support parallel execution of services beside their given ability to express interleaving, decisions and looping. The model has then been used to build the basis for a new type of rich Web Mashups that are responsive the the users' preferences and device context by supporting late binding of 3rd party services during runtime.

Within future work, algorithms for the automatic creation of service compositions are developed. Here, the descriptions for 3rd party services are extended by lightweight semantics to support the automatic creation of workflows and binding of services based on a request given by either a user or another service. Moreover, a GUI is developed that enables the manual creation of service compositions in an intuitive way; this representation is then transformed into corresponding workflow and dataflow graphs automatically.

References

[1] www.programmableweb.com.

[2] R. Alur, C. Courcoubetis, and D. Dill. Model-checking for real-time systems. In *Proceedings of of the 5th Annual Symposium on Logic in Computer Science*, pages 414–425. IEEE Computer Society Press, 1990.

[3] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Business Process Execution Language for Web Services, Version 1.1, May 2003. [Online]. Available: http://www.ibm.com/developerworks/library/specification/ws-bpel/. [Accessed: November, 2007].

[4] J. Berstel. *Transductions and Context-free Languages*. B.G. Teubner, Stuttgart, 1979.

[5] A. Bucchiarone and S. Gnesi. A Survey on Services Composition Languages and Models. In A. Bertolino and A. Polini, editors, *in Proceedings of International Workshop on Web Services Modeling and Testing (WS-MaTe2006)*, pages 51–63, Palermo, Sicily, ITALY, June 9th 2006.

[6] R. Chinnici, J.-J. Moreau, A. Ryman, and S. Weerawarana. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. In *World Wide Web Consortium (W3C) recommendation*, January 2006. [Online]. Available: http://www.w3c.org/TR/wsdl20/. [Accessed: Mar. 24, 2006].

[7] E. M. J. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts and London, England, 1999.

[8] F. Curbera, Y. Goland, J. Klein, F. Leymann, D. Roller, S. Thatte, and S. Weerawarana. Business Process Execution Language for Web Services (Version 1.0), July 2002.

[9] D. Dill. Timing assumptions and verification of finite-state concurrent systems. In J. Sifakis, editor, *Proceedings of the International Workshop on Automatic Verification Methods for Finite State Systems*, number 407 in LNCS, pages 197–212. Springer, 1989.

[10] L. Doldi. *UML2 illstrated, Developing Real-Time & Communication Systems*. TMSO, 2003.

[11] J. Duhl. Rich Internet Applications. *Whitepaper (sponsored by Macromedia and Intel)*, page Online: http://www.adobe.com/resources/ business/rich_internet_apps/whitepapers.html, 2003.

[12] S. Eilenberg. *Automata, Languages, and Machines*, volume Volume A. Academic Press, New York, 1974.

[13] R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[14] M. J. Hadley. Web application description language (wadl). Technical report, Sun Microsystems Inc., November 2006.

[15] M. Huth and M. Ryan. *Logic in Computer Science - Modeling and reasoning about systems*. Cambridge University Press, 2004.

[16] M. Jaeger, L. Engel, and K. Geihs. A Methodology for Developing OWL-S Descriptions. In *First International Conference on Interoperability of Enterprise Software and Applications Workshop on Web Services and Interoperability (INTEROP-ESA '05)*. Springer, 2005.

[17] F. Lautenbacher and B. Bauer. A Survey on Workflow Annotation & Composition Approaches. In *Proceedings of the*

Workshop on Semantic Business Process and Product Lifecycle Management (SemBPM) in the context of the European Semantic Web Conference (ESWC), pages 12–23, Insbruck, Austria, 7th June 2007.

[18] G. Lawton. New ways to build rich internet applications. *Computer*, 41(8):10–12, Aug. 2008.

[19] F. Leymann. Web Service Flow Language (WSFL 1.0). In *IBM*, May 2001.

[20] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, 1992.

[21] E. Ort, S. Brydon, and M. Basler. Mashup styles, part 1: Server-side mashups. Technical report, Sun Developer Network (SDN), May 2007.

[22] E. Ort, S. Brydon, and M. Basler. Mashup styles, part 2: Client-side mashups. Technical report, Sun Developer Network (SDN), August 2007.

[23] C. E. Ortitz. Introduction to J2ME Web Services, April 2005. [Online]. Available: http://developers.sun.com/mobility/apis/articles/wsa. [Accessed: October, 2007].

[24] H. Pfeffer. UPPAAL Model Checking as Performance Evaluation Technique. Master's thesis, Rheinische Friedrich-Whilhelms-Universität Bonn, 2005.

[25] H. Pfeffer, L. Bassbouss, and S. Steglich. Structured service composition execution for mobile web applications. In *Proceedings of the 12th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS 2008)*, volume ISBN: 978-0-7695-3377-3, pages 112–118, Kunming, China, 2008. IEEE Computer Society Press.

[26] H. Pfeffer, S. Kr"ussel, and S. Steglich. Fuzzy Service Composition Evaluation In Distributed Environments. In *In Proceedings of I-CENTRIC 2008*, 2008.

[27] H. Pfeffer, D. Linner, C. Jacob, and S. Steglich. Towards Light-weight Semantic Descriptions for Decentralized Service-oriented Systems. In *Proceedings of the 1st IEEE International Conference on Semantic Computing (ICSC 2007)*, volume CD-ROM, Irvine, California, USA, 17-19 September 2007. PLJ+07.

[28] H. Pfeffer, D. Linner, and S. Steglich. Dynamic adaptation of workflow based service compositions. In *ICIC '08: Proceedings of the 4th international conference on Intelligent Computing*, number ISBN: 978-3-540-87440-9, pages 763–774, Berlin, Heidelberg, 2008. Springer-Verlag.

[29] H. Pfeffer, D. Linner, and S. Steglich. Modeling and controlling dynamic service compositions. *Computing in the Global Information Technology, 2008. ICCGI '08. The Third International Multi-Conference on*, pages 210–216, 27 2008-Aug. 1 2008.

[30] S. Thatte. XLANG - Web Services for Business Process Design, 2001.

[31] The OWL Services Coalition. OWL-S: Semantic Markup for Web Services, November 2004. [Online]. Available: http://www.daml.org/services/owls/1.1/. [Accessed: February 26, 2007].

[32] U. University and A. University. UPPAAL 4.0.6. http://www.uppaal.com, December 2007.

[33] W. van der Aalst. Don't go with the flow: Web services composition standards exposed. *IEEE Intelligent Systems*, 2003.

A Method for Automatically Eliciting node Weights in a Hierarchical Knowledge-Based Structure for Reasoning with Uncertainty

S. E. Hegazy, C. D. Buckingham
School of Engineering and Applied Science, Aston University, Birmingham, UK.
hegazys@aston.ac.uk c.d.buckingham@aston.ac.uk

***Abstract* - Hierarchical knowledge structures are frequently used within clinical decision support systems as part of the model for generating intelligent advice. The nodes in the hierarchy inevitably have varying influence on the decision-making processes, which needs to be reflected by parameters. If the model has been elicited from human experts, it is not feasible to ask them to estimate the parameters because there will be so many in even moderately-sized structures. This paper describes how the parameters could be obtained from data instead, using only a small number of cases.**

The original method [1] is applied to a particular web-based clinical decision support system called GRiST, which uses its hierarchical knowledge to quantify the risks associated with mental-health problems. The knowledge was elicited from multidisciplinary mental-health practitioners but the tree has several thousand nodes, all requiring an estimation of their relative influence on the assessment process. The method described in the paper shows how they can be obtained from about 200 cases instead. It greatly reduces the experts' elicitation tasks and has the potential for being generalised to similar knowledge-engineering domains where relative weightings of node siblings are part of the parameter space.

Keywords: *Clinical Decision Support Systems; Mental Health; Risk Screening; Hierarchical Knowledge; Decision Trees; Mathematical Modelling.*

I. INTRODUCTION

Clinical decision support systems (CDSSs) often work in complex domains that require modelling of human expert knowledge [2,3]. The resulting models may possess high numbers of parameters that need to be instantiated, which is extremely time-consuming for the domain experts and may not even be realistically achievable. An important element of human expertise is its hierarchical structuring [2], which leads to equivalent knowledge structures within CDSSs. These structures or trees have many nodes and the influence of each child node on its parent node will vary across the siblings when it comes to processing uncertainty through the tree. Each node will therefore require a parameter to represent its particular influence on the decision making process, which adds up to a very large number of values to be given by the domain experts on whom the CDSS is being modelled. This paper describes a method for inducing the parameters from a small number of cases instead and shows how it has been applied to a particular CDSS in the domain of mental health risk assessment. The method has the potential for being generalised to any tree where siblings of single parent nodes need individual weights to fit the data. The paper will begin by introducing the domain and the specific CDSS.

A. Risk assessment in mental health

Risk screening in the mental health field is a particularly complex procedure but lacks much assistance beyond paper-based tools [4]. At present, actuarial approaches to risk prediction gain favour because of their evidence base, but have a predictive value that remains unsatisfactory. They also tend to rely on isolated factors, not combinations [5], and ignore the individual qualitative and idiosyncratic patient data that support clinical judgements in practice [6]. There is a need for tools based on clinical expertise as well as empirical evidence and this was precisely the motivation for developing the Galatean Risk Screening Tool, GRiST [7, 8]. It is a web-based CDSS that is designed to assist the early detection of multiple risks, including suicide, self-harm, harm to others, self-neglect, and vulnerability amongst people with mental health problems. It is the only risk-assessment tool that uses a computational model of psychological processes to represent structured clinical judgements of multidisciplinary mental-health practitioners [9, 10].

GRiST has successfully elicited the hierarchical knowledge used by expert mental-health practitioners [11] but it generated a tree with over one thousand nodes, each of which has a parameter representing its relative influence on the assessment process. Asking the domain experts to set these parameters was not feasible and an alternative approach was investigated instead.

In essence, GRiST is a weighted decision tree where risk is represented by fuzzy-set membership grades (MGs) [12] that are associated with each node of the tree. Figure 1 shows a small portion of the GRiST tree for

suicide risk. The bottom level boxes are the data for a patient assessment (case). These generate a MG at the matching leaf node using a function that depends on some parameters given by the experts for each leaf node (see [9] for more details). The MGs then propagate up the risk hierarchy and eventually to the top level risks, where the MG associated with a risk represents the simulated clinical risk judgement. The relative influence (RI) of each node in the hierarchy is a parameter that decides how much risk is propagated up the tree by a node compared to its siblings [9]. This parameter also needs to be set so that it reflects the expertise of mental-health practitioners. Getting them to do it themselves as part of the knowledge elicitation process is an arduous task when the tree has so many nodes. This makes it unlikely that a large enough set of participants can be obtained to ensure the consensus for each RI is reliable, as opposed to eliciting the leaf node parameters, which are far fewer: 192 for GRiST.

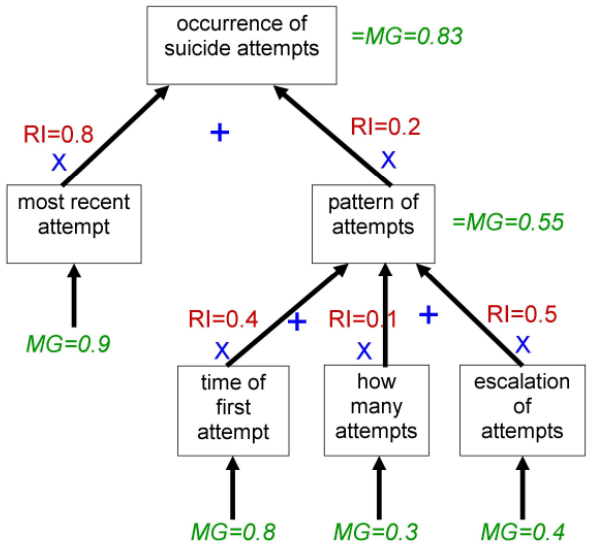


Figure 1: *A portion of the GRiST for suicide risk showing how the relative influences of the nodes moderate the flow of risk. Each node MG is multiplied by its associated RI and summed with the siblings to give the parent MG. Note that the actual values are hypothetical.*

In this paper, we devise an algorithm that induces the RIs from the clinical judgements given by expert mental-health practitioners for patient cases. This will mean the RIs are modelled on the clinicians' own risk judgements because the RIs are set to the exact values required for simulating those judgements. It depends on knowing the MGs at the leaf nodes for a patient's data along with the associated clinical risk judgements, where the risk judgements equate to the MG that GRiST needs to generate at the root node (risk) for that patient. The number of cases required to solve the RIs must be the same as the number of cues in the patient's data set. For GRiST, these judgements are given by clinicians as part of their everyday use of GRiST in practice. Hence the elicitation process has been reduced to providing only the parameters for the 192 leaf nodes. It is important, if not mandatory, for having such an automated system to elicit RIs because the sheer number is likely to mean experts don't do it accurately themselves.

This paper will give some background to the basic problem, after which the method and algorithm will be described. It will conclude with a discussion about how the approach could have generic applicability and be extended.

II. BACKGROUND

The problem we are trying to solve could be represented in a more generalized form, which is a decision tree with weighted inputs. Each input at the leaves contributes to the final decision at the top of the tree, through a weight that determines how much influence the node has compared to its siblings. Every node has these weights applied to its child nodes and, for GRiST, there is an additional constraint that the weights across all the sibling nodes must sum to unity. The task is to find a way of automatically deducing the weights throughout the tree from a minimal set of inputs and outputs.

Most algorithms that have been developed for learning decision trees are variations on a core algorithm that employs a top-down, greedy search through the space of possible trees. These algorithms generally construct a decision tree, *T*, from a set of training cases [13]. J. Ross Quinlan developed the first algorithm, ID3 [14], and based it on the Concept Learning System (CLS) algorithm [15]. Other methods like CART (Classification and Regression Trees) were introduced for the induction of a tree [16].

Variations on the above methods usually deal with the type of the input variables, the data pool or set properties, or the output type (i.e. continuous or discrete data) [17-19]. Most of these methods attempt to construct the tree without prior knowledge of the desired tree structure. This means, they try to predict the layout of the tree and number of nodes based on the training cases. The trees are then pruned and optimized to the minimum structure that satisfies the classes in the training instances.

Our problem is very different. We aim to model the GRiST decision tree parameters mathematically, since the structure of the tree is known in advance from the psychological model that has been induced from the experts [10, 11]. Hence, we are in control of the structure, don't require pruning and optimization

processes, and can use the training sets purely to induce the unknown weights in our model.

III. METHODOLOGY

In this section, we introduce the general structure of the decision tree used by GRiST, which is the same structure that our model will use to calculate the RI values. It is shown in *Figure 2* as follows:

L_{Rn} : denotes the RIs in level n.
M_n : denotes the MGs (Membership Grades) in level n.
M_{xy} : denotes the MG of node y in level x; y=0 to Zjh , where Zjh is the number of children of node number h-1 at level j.
R_{ti} : denotes the RI of node number i at level t on the total MG at level t-1 which equals $M_{(t-1)y}$ where y is the number of the parent node of R_{ti} .

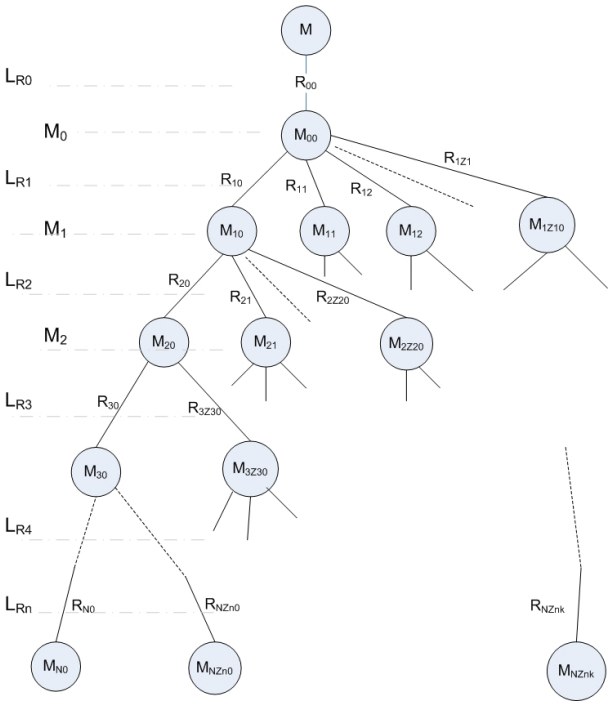


Figure 2: *The General GRiST DSS Tree.*

To find M, the total membership grade of the tree (which represents the overall diagnosis or risk of the patient's mental health [9]), there are several methodologies we could follow. One would be to train the model using known cases and, assuming that leaf MG values and M are given, we could use a neural networks simulation. The problem with neural networks is, though, that we won't be able to represent the internal hierarchical structure of the GRiST tree as given by the experts, which is crucial to the explanation of how risks are generated. We have thus developed a method that maintains the tree structure within a mathematical representation and uses training sets to induce the values of RIs. To model the tree mathematically, we follow the psychological model underlying GRiST [9], which defines how to calculate the overall result, M (the details of the model are not relevant for this paper because our algorithm applies to the RIs only, not the generation of MGs at the leaf nodes). The MG at each node is the summed product of each child node's MG and RI, which then feeds through to the next parent node in the same way, as follows:

$$M = R_{00}\, M_{00}$$

$R_{00} = 1$, thus, $M = M_{00}$

$$M = R_{00}\,(R_{10}\,M_{10} + R_{11}\,M_{11} + R_{12}\,M_{12} + \ldots\ldots + R_{1Z10}\,M_{1Z10})$$

If we expand the calculations for the MGs in the child nodes, we get

$$M = R_{00}\,(R_{10}\,(R_{20}\,M_{20} + R_{21}\,M_{21} + R_{22}\,M_{22} + \ldots\ldots + R_{2Z10}\,M_{2Z20}) + \ldots\ldots .. R_{1Z10}\,(\ldots\ldots\ldots\ldots\ldots\ldots..))$$

If we continue this process, until we reach the leaves, the resulting expression will be the sum of the products of all RIs along the path to a leaf node and that leaf node's MG, which creates a certain pattern for the multiplication expression that we will clarify and make use of later.

To illustrate the above, we use a simpler example of a tree with just two levels, as shown in Figure 3, where a to g are used to represent the specific leaf node MGs of M_{20} to M_{27} for clarity.

$$M = R_{00}\,(R_{10}\,M_{10} + R_{11}\,M_{11} + R_{12}\,M_{12})$$
$$= R_{00}\,(R_{10}\,(R_{20}\,a + R_{21}\,b) + R_{11}\,(R_{22}\,c + R_{23}\,d) + R_{12}\,(R_{24}\,e + R_{25}\,f + R_{26}\,g) \quad (1a)$$

Or:

$$M = R_{00}\,R_{10}\,R_{20}\,a + R_{00}\,R_{10}\,R_{21}\,b + R_{00}\,R_{11}\,R_{22}\,c + R_{00}\,R_{11}\,R_{23}\,d + R_{00}\,R_{12}\,R_{24}\,e + R_{00}\,R_{12}\,R_{25}\,f + R_{00}\,R_{12}\,R_{26}\,g \quad (1b)$$

Since, a to g are given, the unknowns are all the Rs. The top-level M is also given, because it represents the clinical judgement associated with the case (for different cases, we will use M1, M2, M3, …). We have several of the above equations, one per case, and can regard them as a system of linear simultaneous equations. To solve the R

values, we need the same number of cases as there are leaf nodes.

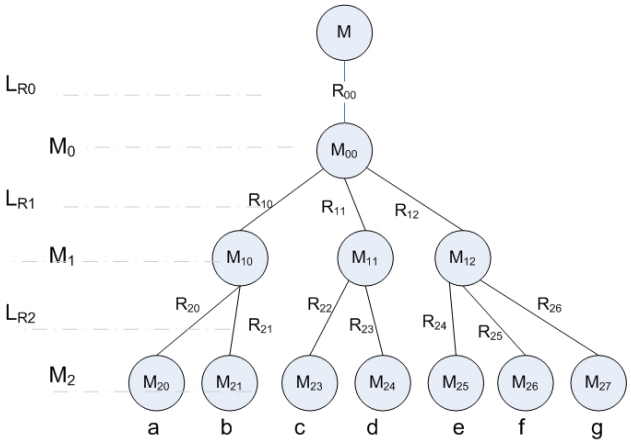


Figure 3: *A simple two level GRiST tree.*

To simplify, we rename RI products along a path as:

$$A = R_{00} R_{10} R_{20}$$
$$B = R_{00} R_{10} R_{21}$$
$$C = R_{00} R_{11} R_{22}$$
$$D = R_{00} R_{11} R_{23}$$
$$E = R_{00} R_{12} R_{24}$$
$$F = R_{00} R_{12} R_{25}$$
$$G = R_{00} R_{12} R_{26} \quad (2)$$

to give seven equations for our example with $R_{00} = 1$ (from the RI properties).

The system can be set up as a set of linear simultaneous equations, as follows:

$$M1 = a1.A + b1.B + c1.C + d1.D + e1.E + f1.F + g1.G$$

$$M2 = a2.A + b2.B + c2.C + d2.D + e2.E + f2.F + g2.G$$

.... and so on ....

$$M7 = a7.A + b7.B + c7.C + d7.D + e7.E + f7.F + g7.G \quad (3)$$

Solving (3) is straightforward (using matrices), which gives us A to G. But originally, we had eleven unknowns, so to determine RIs, we need an extra four equations in addition to the above seven. For this we use the inherent property of RIs that they must sum to one across all siblings:

$$\sum_{y=0}^{Zxn} R_{xy} = 1 \quad (4)$$

In our case this gives us:

$$R_{10} + R_{11} + R_{12} = 1 \quad (4b)$$
$$R_{20} + R_{21} = 1$$
$$R_{22} + R_{23} = 1$$
$$R_{24} + R_{25} + R_{26} = 1 \quad (5)$$

So we have eleven equations and eleven unknowns.
By substitution, we can solve the system exploiting another pattern:

$$A / B = (R_{10} . R_{20}) / (R_{10} . R_{21})$$

$$= R_{20} / R_{21}$$

So: $R_{21} = (B / A) R_{20}$ (5a)

Substituting in the relevant equation, we get:

$$R_{20} + R_{21} = R_{20} + (B / A) R_{20} = 1$$

Or: $R_{20} (1 + (B/A)) = 1$

Or: : $R_{20} ((A+B) / A) = 1$

Thus: $R_{20} = A / (A+B)$

By continuing in the same manner, we can obtain the rest of the RIs.

$$R_{20} = \frac{A}{(A+B)}$$
$$R_{21} = \frac{B}{(A+B)}$$
$$R_{22} = \frac{C}{(C+D)}$$
$$R_{23} = \frac{D}{(C+D)}$$
$$R_{24} = \frac{E}{(E+F+G)}$$
$$R_{25} = \frac{F}{(E+F+G)}$$
$$R_{26} = \frac{G}{(E+F+G)} \quad (5b)$$

In other words, each leaf RI can be found as a function of the RI products along the path from each sibling leaf to the root node. These products, A to G, have been solved

from the simultaneous equations, so each individual leaf RI can thus be calculated.

IV. THE COMPLETE ALGORITHM

The input to the algorithm would be n vectors of known and diagnosed cases given by experts. In the example for Figure 3, that vector will contain the following:

$$V = (M, a, b, c , d, e, f, g) \quad (6)$$

where M is the top-level clinical judgement given by the clinician for the patient MGs of a,b, … g (i.e. the leaf-node MGs generated directly from the patient values).

The algorithm we propose can be divided into two steps: solving for the multipliers of each leaf MG, which are the products of the RIs along the path from the leaf to the root (i.e. A to G), and then solving for the individual RIs themselves.

Step 1: Solving for Multipliers

The first step will be solving n simultaneous linear equations, where n is the total number of leaves of the GRiST tree (seven, a to g, in Figure 3):

M1 = a1. A + b1. B + c1. C + d1. D + e1. E + f1. F + g1 .G
M2 = a2. A + b2. B + c2 .C + d2 . D + e2. E + f2 .F + g2. G
……
……
M7 = a7. A + b7. B + c7. C + d7. D + e7. E + f7. F + g7 .G

(7)

Or in matrix form:

$$\begin{pmatrix} M1 \\ M2 \\ M3 \\ M4 \\ M5 \\ M6 \\ M7 \end{pmatrix} = \begin{pmatrix} a1 & b1 & c1 & d1 & e1 & f1 & g1 \\ a2 & b2 & c2 & d2 & e2 & f2 & g2 \\ a3 & b3 & c3 & d3 & e3 & f3 & g3 \\ a4 & b4 & c4 & d4 & e4 & f4 & g4 \\ a5 & b5 & c5 & d5 & e5 & f5 & g5 \\ a6 & b6 & c6 & d6 & e6 & f6 & g6 \\ a7 & b7 & c7 & d7 & e7 & f7 & g7 \end{pmatrix} \times \begin{pmatrix} A \\ B \\ C \\ D \\ E \\ F \\ G \end{pmatrix} \quad (8)$$

Equation 8 can be solved using Gaussian Elimination so we now know the values of A to G, which is given by the solution, S:

$$S = (A, B, C, D, E, F, G) \quad (9)$$

Step 2: Solving for individual RIs

To find each RI, we look at a general leaf node and its children (see *Figure 4).*

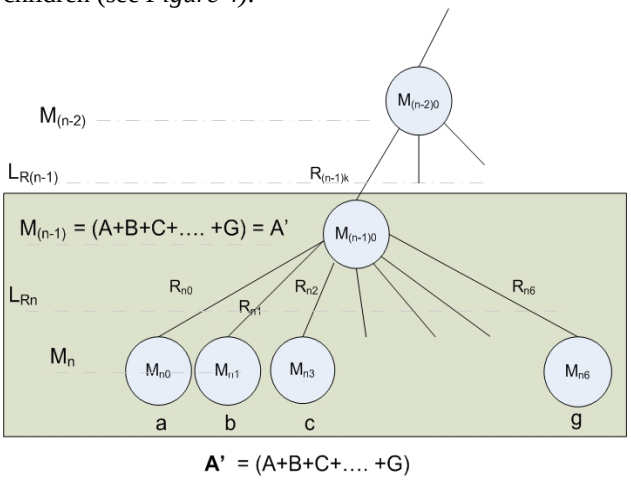


Figure 4: *A general leaf node with seven children.*

The challenge is to devise a systematic way for deriving the solution. Let us take a slice of matrix S, and call it S' for simplicity; it only contains entries for leaf nodes that are siblings and that therefore share the same ancestral path of RIs, which is $R_{10} \ldots R_{(n-1)0}$ in our example.

$$S' = \begin{pmatrix} R_{10} \ldots\ldots R_{(n-1)0} R_{n0} \\ R_{10} \ldots\ldots R_{(n-1)0} R_{n1} \\ R_{10} \ldots\ldots R_{(n-1)0} R_{n2} \\ R_{10} \ldots\ldots R_{(n-1)0} R_{n3} \\ R_{10} \ldots\ldots R_{(n-1)0} R_{n4} \\ R_{10} \ldots\ldots R_{(n-1)0} R_{n5} \\ R_{10} \ldots\ldots R_{(n-1)0} R_{n6} \end{pmatrix} \quad (10)$$

From the GRiST model [9], we know that:

$$R_{n0} + R_{n1} + R_{n2} + R_{n3} + R_{n4} + R_{n5} + R_{n6} = 1 \quad (11)$$

We will convert Equation 10 into a function of only one variable, e.g. R_{n0}. To do this we use S', where each of the rows are represented by a symbol, A to G, for the RI product along the path.

$B/A = R_{n1} / R_{n0}$

$$R_{n1} = (B/A) . R_{n0}$$

$C/A = R_{n2} / R_{n0}$

$$R_{n2} = (C/A) . R_{n0}$$

…. and so on ….

$G/A = R_{n6} / R_{n0}$

$$R_{n6} = (G/A) . R_{n0}$$

Substituting in Equation 11:

$R_{n0} + (B/A) . R_{n0} + (C/A) . R_{n0} + (D/A) . R_{n0} + (E/A) . R_{n0} + (F/A) . R_{n0} + (G/A) . R_{n0} = 1$

Factoring out R_{n0}:

We get

$$R_{n0} = \left(\frac{A}{A+B+C+D+E+F+G} \right)$$

Solving in the same way, we obtain:

$$R_{n1} = \left(\frac{B}{A+B+C+D+E+F+G} \right)$$

… and so on to …

$$R_{n6} = \left(\frac{G}{A+B+C+D+E+F+G} \right) \quad (12a)$$

Hence the general rule in the algorithm, to find a certain RI in the leaf nodes is:

$$RI_j = \left(\frac{S'(j)}{\sum_{j=1}^{k} S'(j)} \right) \quad (12b)$$

Where j is the leaf node MG (in our example, a,b,c, …), k is the total number of siblings, and S' is the product of all RIs along the path from the specified leaf node to the root node..

Step 3: Shrinking the tree

Having found the RIs of the leaf node (see Figure 4), we can now calculate the MG for the parent node, $M_{(n-1)0}$, which can then become a leaf itself. We can do this for all the parent nodes that have leaf nodes as children and, by converting them into leaves themselves once their MG has been calculated, the tree is shrunk.

Summary of the generalised algorithm

So far, the explanation has used specific trees to illustrate it. We can now generalize the algorithm as follows.

Inputs:

$V1 = (M1, M_{n01}, M_{n11}, ……, M_{nk1})$

To:

$Vk = (Mk, M_{n0k}, M_{n1k}, ……, M_{nkk})$ (16)

Where:
M1 to Mk : are the k different cases outcomes.
M_{n0y} : is the input MG at the leaf on the nth level (lowest level) of the GRiST tree of the y^{th} input vector (Vy).
We need k vectors to solve the resulting k simultaneous equations where k = the number of leaf nodes of the GRiST tree = the number of cases required.

Outputs:

RI values, representing the node weightings for every node in the tree.

Procedure:

Step one:

Solve the following simultaneous equations:

$$\begin{pmatrix} M1 \\ M2 \\ M3 \\ .... \\ ... \\ .... \\ Mk \end{pmatrix} = \begin{pmatrix} M_{n01} & M_{n11} & M_{n21} & ... & ... & .. & M_{nk1} \\ M_{n02} & M_{n12} & M_{n22} & ... & .... & ... & M_{nk2} \\ M_{n03} & M_{n13} & M_{n23} & .... & .... & ... & M_{nk3} \\ ... & .... & .... & .... & ... & ... & ... \\ ... & .... & ... & ... & ... & ... & ... \\ .... & ... & ... & ... & ... & ... & .... \\ M_{n0k} & M_{n1k} & M_{n2k} & ... & ... & ... & M_{nkk} \end{pmatrix} \times \begin{pmatrix} A1 \\ A2 \\ A3 \\ ... \\ ... \\ ... \\ Ak \end{pmatrix} \quad (17)$$

The above matrix is kXk in dimension.
The solution yields vector A1 to Ak.

Step two:

We use S' to denote a sub tree of each node at level (n-1), where n is the deepest level of the tree where all nodes are leaf nodes.
Hence we have: S'1 to S'h where h is the number of nodes at level (n-1) in the GRiST tree.
For each subtree, S'j, we solve to find its RIs.

$$RI_{nr} = \left(\frac{S'j(r)}{\sum_{r} S'j(r)} \right) \qquad (18)$$

Where *r* represents the children (and thus leaf nodes) of parent node *j*, with r going from 1 to the number of leaves of node j at level (n - 1); j = 0 to h.

Step three:

Once the RIs have been found at a particular level, the tree can be shrunk by a level by making the parent nodes the new leaf nodes with their MGs calculated by:

$$M_{(n-1)h} = \sum_{r} S'j(MGj) \qquad (19)$$

Once the new shadow MGs are found for the new level, we can go to step two and repeat step two and three for the new tree. This process is continued n times (for an n-level tree). At the end, we will have determined all the RIs in the tree.

Case study:

This part of the paper demonstrates the effectiveness of the algorithms using a case study with arbitrary numbers. We will use our algorithm to calculate the RI values in the tree shown in Figure 5. The tree has six leaves (A to F), hence we need six training cases. The following matrix sets up the synthetic data in the format of Equation 17:

$$\begin{pmatrix} 0.3 \\ 0.4 \\ 0.9 \\ 0.7 \\ 0.8 \\ 0.1 \end{pmatrix} = \begin{pmatrix} 0.1 & 0.3 & 0.4 & 0.2 & 0.3 & 0.6 \\ 0.1 & 0.2 & 0.5 & 0.4 & 0.6 & 0.2 \\ 0.2 & 0.1 & 0.3 & 0.7 & 0.8 & 0.7 \\ 0.3 & 0.4 & 0.5 & 0.6 & 0.7 & 0.3 \\ 0.2 & 0.3 & 0.4 & 0.3 & 0.9 & 0.2 \\ 0.3 & 0.1 & 0.6 & 0.5 & 0.5 & 0.4 \end{pmatrix} \times \begin{pmatrix} A \\ B \\ C \\ D \\ E \\ F \end{pmatrix} \qquad (20)$$

Using Gaussian Elimination, to solve the above matrix for the unknowns, we obtain:

A = -0.44 D = 0.44
B = 0.92 E = 0.964
C = -1.067 F = 0.196

Note that we use 3 decimal points approximation for simplicity (rounding).

.

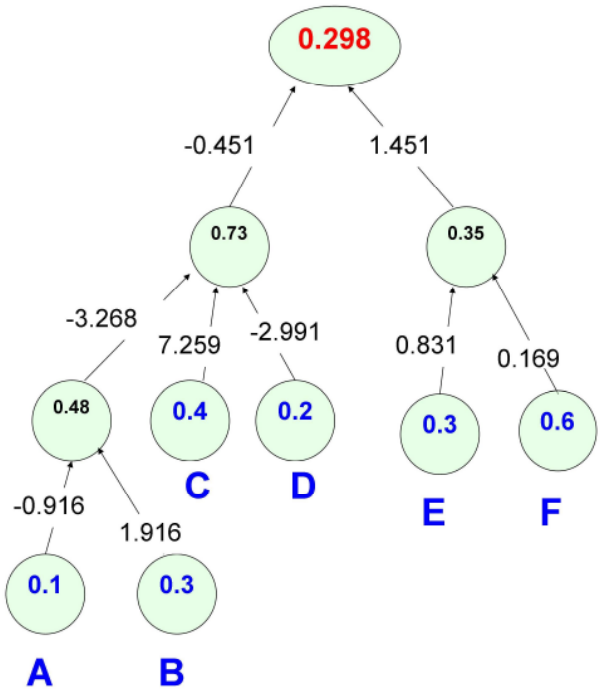


Figure 5: *A sample decision sub-tree.*

Using Equation 18 and the propagation technique in Equation 19, we obtain all the RI values as in Figure 5.

To verify the model, we use the first training case (first line in Equation 20) as an input (on Figure 5, it is the number printed inside each leaf node, A to F). Propagating through the decision tree using the new RI values, we finally reach a decision (M = 0.298, inside the top node). This is almost the same as the desired output in the original test case, in Equation 20 (i.e. 0.3). The error is due to approximation and using only three decimal points precision.

The case study shows that solutions may require negative RI values, which is only a problem if the semantics of the knowledge domain demand positive values. For the GRiST domain, and probably many other knowledge-based systems, the concept of negative RIs is not psychological meaningful, although semantically it can be explained in terms of a bigger span between the RIs of the siblings and those could then be mapped to normalized values. It is possible that real-world data, where clinicians have provided risk assessments for a

given set of patient values, will have inherent constraints that mean the RIs willl not be negative. However, it remains a possibility that limiting RIs to positive values would mean a solution cannot be found. In the next section, we will discuss an extension to the method that will circumvent this problem.

V. CONCLUSION AND FUTURE WORK

In this paper, we have addressed the problem of eliciting parameters in the GRiST tree structure [10, 11]. These parameters can then be used to analyze new cases and provide advice for mental health practitioners. The techniques presented here are extending our ARRIVE [1] algorithm, and provide a robust mathematical calculation of the Relative Influence (RI) values in the GRiST tree [9] that that are crucial for enabling risk quantifications to be generated. Similar approaches could be relevant to many intelligent knowledge-based systems based on human expertise where the knowledge is in a hierarchical structure and the nodes have varying influence on the decision making processes. For GRiST, the RIs represent varying weights of sibling nodes on their parents and were normalised so that the total weighting across siblings was unity for all nodes.

At present, the method is intended to initialise the node weightings from a fixed number of cases equal to the number of leaf nodes in the tree, where the risk judgements have been given by expert clinicians for the set of patient data associated with those leaf nodes. It would be better, though, if the weightings could be incrementally updated as new cases are classified and future work will explore techniques for accomplishing this. It means the RIs would be a more representative consensus for the clinicians, having been induced from an ever-increasing data set. The resulting weights would thus be best estimates from the data and would enable constraints on the range of allowable values to be set without jeopardising the generation of solutions. The method described in this paper could create the initial weights that would then be updated as new cases arrive.

Other aspects of future work include analyzing the sensitivity of the algorithms to variations in patient data as well as the impact of missing data and noise in the learning data sets. An interesting problem is how to determine ways of quantifying error margins and confidence in the risk judgements based on the constitution of patient data sets.

REFERENCES

[1] S. E. Hegazy, C. D. Buckingham (2008). ARRIVE: An Algorithm for Robust Relative Values Elicitation, *Proceedings of the International Conference on Computing in the Global Information Technology*, July 2008, 91-96.

[2] Buchanan, B.G., Davis, R., & Feigenbaum, E.A. (2006). Expert Systems: A perspective from computer science. In: K.A. Ericsson, N. Charness, P. Feltovich and R. Hoffman, Editors, *The Cambridge handbook of expertise and expert performance*, Cambridge University Press, New York (2006), pp. 87-103

[3] Berner, E.S. (2006). *Clinical Decision Support Systems: Theory and Practice* (Ed.), 2nd Ed, Springer: New York.

[4] Hawley, C. J., Littlechild, B., Sivakumaran, T., Sender, H., Gale, T. M. & Wilson, K. J. (2006). Structure and content of risk assessment proformas in mental healthcare. *Journal of Mental Health, 15,* 437-448.

[5] Skegg, K. (2005). Self-harm. Lancet, 366, 1471-1483.

[6] Holdsworth, N., & Dodgson, G. (2003). Could a new Mental Health Act distort clinical judgement? a Bayesian justification of naturalistic reasoning about risk. *Journal of Mental Health*, 12(5), 451-462.

[7] Buckingham, C.D. (2007). Improving mental health risk assessment using web-based decision support. *Health Care Risk Report*, 13(3), 17-18.

[8] GRiST [www.galassify.org/grist] GRiST [updated December 2008; cited 2009 Jan 5th]. Available from www.grist.galassify.org/grist

[9] Buckingham, C.D. (2002). Psychological cue use and implications for a clinical decision support system. *Medical Informatics and the Internet in Medicine, **27(4)**,* 237-251

[10] Buckingham, C. D., Adams, A.E. & Mace, C. (2008). Cues and knowledge structures used by mental-health professionals when making risk assessments. *Journal of Mental Health*, 17(3), 299-314.

[11] Buckingham, C.D., Ahmed, A., & Adams, A.E. (2007). Using XML and XSLT for flexible elicitation of mental-health risk knowledge. *Medical Informatics and the Internet in Medicine*, 32(1), 65-81.

[12] Zadeh LA. Fuzzy sets. Information Control 1965;8:338-53.

[13] Ivan Bratko, Dorian Šuc Learning qualitative models Published in *AI Magazine*, 2003, vol. 24, no. 4, pp. 107-119, © AAAI Press

[14] J. R. Quinlan (1975). *Machine Learning*, vol. 1.

[15] J.R. Quinlan (1986). Induction of Decision Trees, Machine Learning, (1), 81-106

[16] Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and regression trees.* Monterey, CA: Wadsworth & Brooks/Cole Advanced Books & Software

[17] Quinlan. (1992). *C4.5: Programs for Machine Learning*, Morgan Kaufmann

[18] C. Z. Janikow, "Exemplar learning in fuzzy decision trees," *Proc. FUZZIEEE*, pp. 1500–1505, 1996

[19] Harris Drucker and Corinna Cortes. Boosting decision trees. In *Advances in Neural Information Processing Systems 8*, pages 479–485, 1996.

Addressing Data Model Variability and Data Integration within Software Product Lines

Joerg Bartholdt
Siemens AG
Corporate Technology
Architecture, CT SE 2
Otto-Hahn-Ring 6
81739 Munich, Germany
joerg.bartholdt@siemens.com

Roy Oberhauser
Aalen University
Computer Science Dept.
Beethovenstr. 1,
73430 Aalen, Germany
roy.oberhauser@htw-aalen.de

Andreas Rytina
itemis
Agnes-Pockels-Bogen 1
80992 Munich, Germany
andreas.rytina@itemis.de

Abstract

Software Product Line (SPL) engineering is one approach for addressing customization and variability for software products. However, current state-of-the-art often focuses on feature modeling and component variability while insufficiently addressing data model variability difficulties and their associated complexity. Various software qualities, such as correctness, reusability, maintainability, testability, and evolvability, are negatively impacted.

In this article the Approach for Data Model Variability (ADMV) is described which provides a unified and systematic methodology for providing a consistent view to capture data variability in data models. Adapter generation hides and decouples components from superfluous data elements and supports SPL data integration with the potentially multifarious external systems and devices that a SPL may need to consider. An eHealth SPL case study is presented supporting adapter generation with differential data conversion and data integration with medical devices. The results show that with this approach, data model variability and data integration can be effectively addressed and desirable software qualities preserved.

Keywords - Data Modeling; Data Integration; Variability; Software Product Lines; Unified Modeling Language; Model-Driven Software Development

1. Introduction

One approach that promotes the systematic reuse of software components for different but similar software products (typically products in the same domain) is SPL Engineering (SPLE). Typically the commonalities and variability of the products in the product line are captured and then the development is split into domain (commonalities) and application (additional individual features for the final product). Products are then built by integrating the common artifacts (usually a platform) and optionally configuring them with product-specific artifacts [11] [14].

Significant work and various methodologies for domain analysis and variability modeling for SPLs with a focus on features are, for instance, Feature-Oriented Domain Analysis (FODA) methodology [9], FeatuRSEB [8], PuLSE [2] and "the notion of variability" [25]. Typical feature models in SPLs allow for many ($\sim 10^x$) possible permutations. Considering that an artifact may influence the data model (e.g., adds new data or relations), all artifacts must be able to handle multiple data variants, although they themselves make no use of the available differences. Yet the aforementioned methodologies do not sufficiently support and address variability in the data models. The Orthogonal Variability Model (OVM) [14] does go beyond features to addressing variability in artifacts, but is an abstract approach missing a notation that can be used by automation for data models (also known as schemata). While the challenging issue of data model variability has been previously studied under schema integration [13], data conversion, data and metadata heterogeneity, schema evolution, enterprise application integration, etc., a holistic approach for SPLE is absent.

The Approach for Data Model Variability (ADMV) described in this paper provides a unified methodology for SPLE to consistently view and edit the data within the data model, capture the variability, as well as shield artifact developers from extraneous differences. Additionally, constraint checking support for data integration variability in SPLs via views and adapter

generation is considered, expanding on our previous work [1].

To motivate and demonstrate the features of the ADMV, a case study in the medical domain for an eHealth SPL derived from a third party served as the research basis. It is presented in simplified form for this article. In the following section the scenario and solution requirements are presented. In Section 3, the ADMV is presented and then applied in Section 4 to an eHealth SPL scenario to exemplify the approach. Section 5 considers alternative approaches and Section 6 evaluates the solution against qualities. Related work is then discussed in Section 7. A conclusion and future work discussion follows in Section 8.

2. Scenario and requirements

In eHealth, an increasing market demand for integrated medical information systems and solutions exists, with globalization in the market and customization demands spanning national boundaries. The difficulties for developing and supporting such systems become apparent in time-to-market, labor costs, and error-proneness when aligning and supporting the various data models and data integration needed for such systems. To support a variety of markets, an SPL approach allows the medical information platform customer to select arbitrary features as add-ons to the base product, e.g., date-definition, record repository, security, etc. This entails various challenges, among them that the overall product instance-specific data model will change depending on the features selected, and another challenge being the integration requirements with medical devices and other medical systems.

For example, a medical information system shall work in different hospital environments. Patient data are stored in folders representing a single hospital stay ("clinical record"). All documents created during a later hospital stay are stored in a different folder. In another environment (e.g., triggered by the electronic case record (eCR) specification in Germany [19]) a new folder level "case record" is introduced on the top level. Beneath, the structure follows the previously described "clinical record". All clinical records are sorted by a disease code into the different "case record" folders. That way, explicit access can be granted to medical personnel based on the medical issue (an orthopedic physician treating a broken leg would have no or restricted access to the psychological problems of the same patient). The product line shall be applicable in both types of domains.

Another requirement is the integration of various measurement devices for blood pressure, body temperature, etc., see Figure 2.1. The devices deliver semantically comparable values, but in different data formats, different scales (e.g., °C/°F) and different protocols. Nonetheless, the application must be able to manage that data in a consistent way, abstracting from the differences in detail.

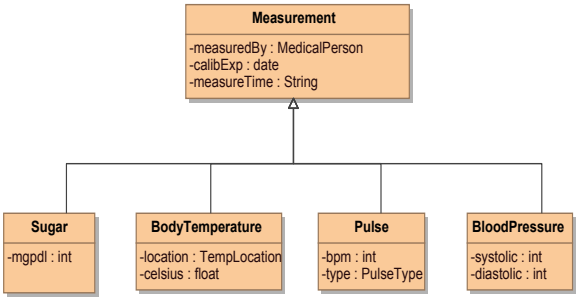


Figure 2.1. UML class diagram of the Measurement data model

A feature of the medical application includes, for instance, the calibration expiration management of the measurement devices. This requires each measurement to carry the information if the measurement was made beyond the calibration expiration and ideally, the expiration date itself (to leave the interpretation to the physician).

Optionally - depending on the environment (e.g., ambulatory vs. stationary), a history of data changes (measured values, patient demographics, etc.) must be recorded, which can be seen as a cross-cutting concern/requirement on domain objects.

Implementing these features and their variability has many effects on the data model of the product instances. E.g., modules for presentation of patient measurements should be programmed with a stable view on the relevant data, ignoring various formats of data delivery (date in `long` or `String` format), data interpretation (°C/°F), and additions like history. This reduces the dependency of such modules on the data model and other components that can vary in the product line instances, thus relieving developers from dealing with this (from their perspective impertinent) variability.

2.1. Requirements

The deficiencies in the examples above illustrate the following requirements that are imposed on the solution to cope with variability in data models:

1) Modeling of the data objects in the solution space must be consistent and provided in a central view (analogous to the feature tree in the problem space that

shows a central view of the variants of the product line). This allows developers and engineers to keep the overview and consistency of the possible product instances and the corresponding data models. The individual products must be derived from this model.

2) Developers of artifacts shall be shielded from the effects of the many possible variants on their code (API and structure of the domain objects) while retaining the compile-time safety that getter/setter navigation in the domain object model guarantees. This includes the demand for loose coupling not only for the functionality of components, but also for the data exchanged between those services.

3) Interoperability of artifacts shall be supported automatically over the SPL lifetime even if the development takes place at different times and disparate locations, thus implying support of multiple versions of the artifacts.

4) In support of correctness, data integrity, data security, and other data-related requirements across the multitude of possible SPL variations, constraints on model consistency and runtime checks shall be supported. Examples are dependency checks of the resulting instance data model (consistency) and authorization constraints for accessing data elements (runtime).

5) Desirable qualities, motivated by SPLE in general, should be supported including consistency, correctness, comprehension, maintainability, usability, efficiency, portability, integration, interoperability, reusability, testability, and traceability.

Although this case study comes from the eHealth domain, the issues are representative and applicable to data variability in SPLs in general.

3. Solution

This section provides a general description of the ADMV process and details on the utilization of fundamental concepts. The approach will then be illustrated by applying the ADVM to an eHealth SPL in Section 4.

3.1 ADMV-Process

The ADMV Process is an UML standards-based approach for SPL data modeling and data integration usable with common Model-Driven Software Development (MDSD) tooling, integrated with feature modeling, and supporting desirable software qualities during SPL development. Unified Modeling Language 2.x (UML2) class diagrams were selected for modeling due to the extensibility via stereotypes (in contrast, e.g., to the entity-relationship diagram) and the plethora of tools available to process the UML model further.

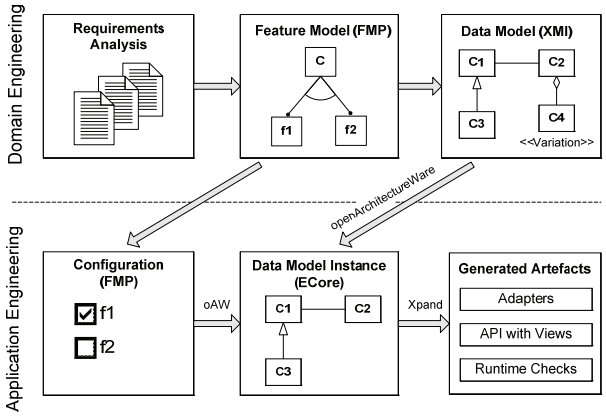


Figure 3.1. ADMV Process

The ADMV process (Figure 3.1) defines several steps in domain engineering and application engineering. These steps are:

1. **Requirements Analysis.** The ADMV starts in the Domain Engineering phase with requirements analysis. Through the analysis of the problem domain, common and variable requirements are collected.
2. **Feature Modeling.** Each variable requirement results in a String which is used as feature name. Dependencies of the features are analyzed and structured in a Feature Model (e.g., using FMP [23]).
3. **Data Modeling.** A Data Model is created in UML2 XMI (XML Metadata Interchange) [30] that includes variations. The first step before integrating variability is the definition of all the common parts. Then, for each feature, the variation points and variants are identified. Eventually the variants are associated with the variation points in connection with an adequate variability type. The ADMV addresses three types of variability: positive - adding new fields, data or relations to the core model; negative - eliminating fields, data, or relations from the core model; and structural - varying the type, cardinality, or naming of elements.
4. **Configuration.** At the start of the Application Engineering phase, a product configuration is created, e.g., in FMP.
5. **Artifact Generation.** Product artifacts are generated such as adapters, converters, views and runtime checks. To accomplish this, the current ADMV Generator implementation uses the

configuration in FMP as well as the Data Model as inputs (e.g., using openArchitectureWare (oAW) [12]) to create a Data Model Instance based on the ADMV metamodel (e.g., an Ecore metamodel [26]), from which the required code artifacts are generated (e.g., using Xpand, oAW's template language for code generation).

6. **Generated Artifact Customization.** Complex Conversions which the ADMV Generator cannot automatically resolve are implemented manually. In addition, the exceptions for the generated runtime checks are implemented manually to fulfill certain tasks when a runtime check fails.
7. **Artifact Integration.** Artifacts are integrated into the build of the data layer and other components.

3.2 Variability types

Negative variability. Negative variability starts from a maximal description (e.g., a UML (Unified Modeling Language) model containing all possible elements of the product line) and deletes the elements that are not connected to selected features. By this reduction, the final model of the selected product instance will be the result. Thus the complete model can be viewed, which may be advantageous if a product instance usually consists of mostly selected features such that the resulting model is close to the complete model (the delta to the complete model is small). On the other hand, it might result in information overload - especially if the product instances consist of only a few selected features such that the resulting models are small (the delta to the complete model is large).

Depending on the selected features, model elements can be removed to derive different product instances. This is reflected in the data model by tagging the different types with the stereotype <<Variation>>. The condition for which it is generated for the product instance is defined by the tagged value {feature = "any feature condition"}. This indicates to the generation process that the elements associated with the feature condition are only generated if the condition evaluates to true, otherwise they are removed.

This is called negative variability since the starting point is a superset of the data model definition and the unnecessary elements are stripped away according to the features selected.

Positive variability. In contrast to negative variability, positive variability starts from a minimal description (a core model, containing only the common parts) and, depending on selected features, additional elements (classes/members/associations) are added to the core model. The parts to merge are described in different places, which may make comprehension of the overall model difficult. This is especially true if there are many additional parts, which is often the case in non-trivial product lines.

Positive variability is useful if cross-cutting concerns should be modeled that cannot be effectively modeled by common base classes and negative variability. As this approach separates the data definition (class plus cross-cutting concerns described outside the class), it contradicts Requirement 1 in Section 2.1. The necessity and benefits in certain circumstances may be reasonable, but we recommend the technique be applied rarely, e.g., due to its potential negative effect on understandability. One technique for applying positive variability in an efficient way is described in [18].

Structural variability. Structural variability describes a change in the model dependent on some feature selection. The element is already contained in the model, but its structure (type, cardinality, association) may vary. Structurally changing the data model is achieved by adding the stereotype <<modify>> to the elements that should be structurally changed and by setting predefined tagged values. Possible tagged values are, e.g., feature, type, cardinality, name and initialValue.

In the resulting data model, the corresponding property is changed. This can also be used to redirect associations by changing the type of the association. An example is given in Section 4 regarding the introduction of additional folder structures due to the electronic case record (eCR) feature.

3.3 Check-Constraints

Constraints are a common concept in modeling and many approaches exist, for instance the Object Constraint Language (OCL). Constraints are used in many different ways: for consistency checks, such as the model itself (e.g., cardinality); for runtime checks (valid references, consistent instantiations); or for optimization [28].

Constraint checking and their languages are a known and powerful capability in assuring modeling correctness, which is especially important when supporting data model variability in a SPL. The ADMV encourages the application of constraint capabilities at the most appropriate points across the tools used in the process. For instance, feature modeling constraints can be utilized to determine the validity of a certain combination of features; data modeling constraints can be applied using active validation (e.g., via OCL or binaries as available in some UML 2 modeling tools) before transformation; transformation constraints can be applied to check conditions (e.g., ensuring that the domain and feature

model are not inconsistent with each other) before or during the generation process; and runtime checks can be automatically included by generators. Thus preconditions, invariants, and postconditions can be specified and carried through the process and applied at the appropriate points.

Consistency checks. The ADMV applies a variety of checks at different phases of the process to ensure the consistency of the models. At modeling time a UML2 tool applies constraints to check the validity of the data model. The uniqueness of members within a class is an example.

To ensure the consistency between the feature model and the data model, additional modeling time checks can be defined. When associating model elements to features, the checks can ensure that the feature model also contains these features.

Aside from ensuring model consistency, the model transformation for deriving product line instances must also be checked. This is done during the generation phase by applying, e.g., oAW-Checks. To ensure that the transformation was correct, oAW-Checks can test if the respective variation points are bound to variants and if the resulting data model is still valid after transformation.

Accessor constraints. To support the verification of certain conditions at run time, the ADMV additionally extends the support of the definition of constraint checks for accessor methods (also known as getters and setters). These constraints can be expressed with a constraint language such as oAW Check. The ADMV Generator transforms these constraints into methods which implement the oAW Check constraints. Every time a getter or setter is called, the associated constraints are successively executed. If one check fails, a runtime exception will be thrown. If all constraints are evaluated to true, the accessor method will be executed.

The analysis of the constraint-string is currently done in the ADMV implementation by the Xtend Parser which is part of the oAW framework. The Xtend Parser returns an abstract syntax tree (AST) which is the input for the ADMV Generator.

3.4 Views

View concepts are known from database systems, model-driven approaches, etc. The way views are considered in the ADMV is from the perspective of the view that a product-line component has on the data model. Certain components may be interested in viewing only parts of entities and shall be shielded from their further development because those components are considered stable and should not have to be adjusted just because the product-line data model changes.

A view is defined as a variant of an entity, which might be shared among several product line instances, or is specific to only one of these instances. An entity can have many views, each of them defining a set of child elements. All child elements have several attributes such as name, type, cardinality, etc. The definition of the view is done in the data model. For each entity there is exactly one complete view (which is the only one potentially persisted in a database) and an unlimited number of projected views. The complete view can be converted to any other projected view and vice versa. The child elements of the projected view can be arbitrarily filled with the source data. In this way it is possible to distribute the content of the source element over multiple elements of the projected view. Vice versa, it is possible to join the source element's data and assign it to a single element of the projected view. In addition, the datatype or properties of the target element can be different from the source element. Thus the structure and content of a projected view and the complete view can be disparate.

Functional components should be shielded from any differences in the data models, which can be achieved with adapters. However, manually written adapters place an additional burden on the developer: besides the initial development, they must be kept consistent with the changes in the data model over time. The ADMV models those adapters together with the data model and generates the code normally automatically – at least for members with the same name. For more difficult conversions, only the getter and setter are generated – the implementation must be added manually. To preserve manual code upon a later update of the data model with subsequent re-generation of the source-code, the Generation Gap pattern [27] may be applied. This is a step towards a consistent view on the data model over the whole SPL over time and it allows the exchange of data between components with different views on those artifacts.

Introducing "Views" gives those types of components a stable, reduced view on the data model. The actual designers and programmers need not be concerned about a variation; they are shielded by their view of an entity.

Note that if modules execute write access to the data, a reverse mapping from the projected view to the complete view must be defined.

Adapters. Adapters are based on the original data object of the product instance and provide a more stable view on the data for components that only require a subset. The adapters provide multiple data views to components and utilize a common data model,

thus conversions are required at runtime. In Figure 3.2a the conversion relationships between views to support the differing projected views desired by components are illustrated, with a maximum number of unidirectional converters required being n(n -1) where n is the number of views required. The maximum number of converters required can be reduced to a linear 2(n-1) if the direct conversions between projected views are avoided and only conversions to and from a complete view are utilized (a star topology) (see Fig 3.2b). This incurs a higher runtime cost of two conversions (once to the common view and then to the desired view) vs. only one, but benefits maintenance and evolution due to the reduced number of conversion methods. The runtime impact is dependent on the number of elements and complexity of conversion.

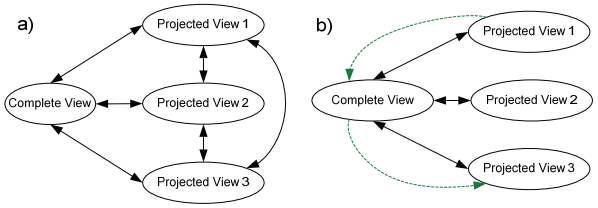


Figure 3.2. Projected view conversions

Data integration via adapters. The view concept supports integration in that it makes different formats and semantics explicit and generates the different adapters.

Advantages of this approach versus manual adapter implementation include the management of data structures from a single unified model, the retention and utilization of the core data model and its variability information by generators when conversion code is generated (the generators use the variability), and the automatic generation of conversion skeleton code and trivial body code (for simple conversions).

3.5 Artifact generation

The process of artifact generation is shown in the Figure 3.3. The data model and the configuration model are the input of the ADMV Generator. While any realization could be used, the current implementation is now described.

The two models are transformed by the template "models2Ecore" to a new data model which is based on an Ecore metamodel. The Ecore-based metamodel is less complex than the UML2 metamodel, making it is easier to define templates for transformation and generation. Initially the variability is not bound in the Ecore data model. It will be bound by the template "toProductModel". This is a model-to-model transformation where all variation points are bound to the configured variants, creating the data model for the configuration. The derived data model is the input for code generation. The views, adapters, and runtime checks are generated by Xpand.

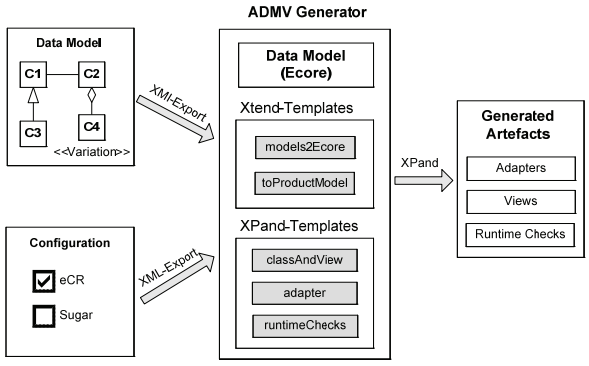


Figure 3.3. ADMV generation process

4. eHealth SPL example

Based on the scenario described in Section 2, the ADMV will now be illustrated.

After requirements analysis (Step 1), a feature model will be defined from the collected features (Step 2). This is the foundation for the product Configuration, defining how features can be combined during the configuration. Figure 4.1 shows the (reduced) Feature Model (FM) for the example domain using the Czarnecki-Eisenecker notation [4]. Hollow circles describe optional features, hollow arcs describe alternative features and filled arcs describe an "or"-relation (select one or more of associated features). A simplified form is used here, e.g., containing functional and non-functional features without explicit constraints, to show the possibilities of the ADMV.

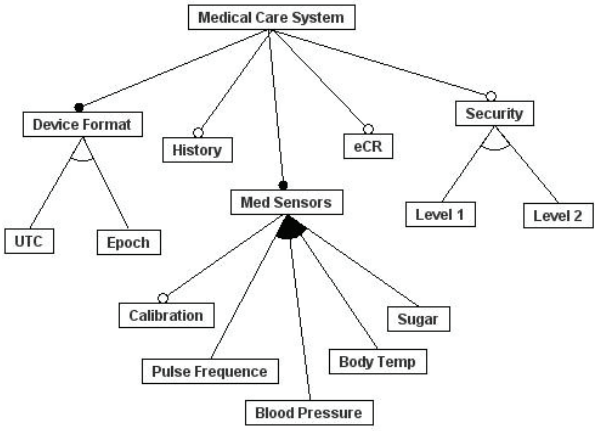


Figure 4.1. Feature model

Four topics were chosen to illustrate the approach: the change in the folder structure due to support of the eCR capability; support for various medical sensors and their different reporting formats (e.g. temperature in °C or °F); security in the sense of authorization of access to data; and a history of data changes. Different variability types can now be chosen to translate the related features into the solution space.

Figure 4.2 shows the FMP representation of the feature model from Figure 4.1 with an example product configuration (Step 4).

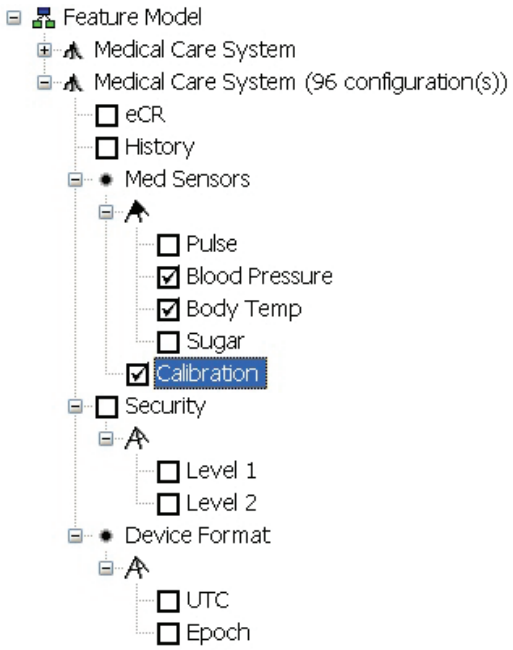


Figure 4.2. Feature model instance

Negative variability (Step 3)**.** Variation points that will be bound through negative variability are marked with the stereotype `<<Variation>>` on class, association, or member level. An appended condition (using bracket: {}) describes which features in the feature model must be selected in order for this part to appear in the data model of the product instance. Note that Boolean expressions are allowed, e.g., `Feature1 AND NOT Feature2`.

Negative variability is shown here to adapt the maximized data model to the resulting instance data model. Figure 4.3 shows the reduced data model.

The presented variation points are the four subclasses and the member `calibExp` (calibration expiration) of the superclass `Measurement`. Because Pulse and Sugar were not selected, the resulting data model shown in Figure 4.4 only contains the measurement types `BloodPressure` and `BodyTemperature`. Because the feature Calibration is selected, the resulting data model contains the member `calibExp`.

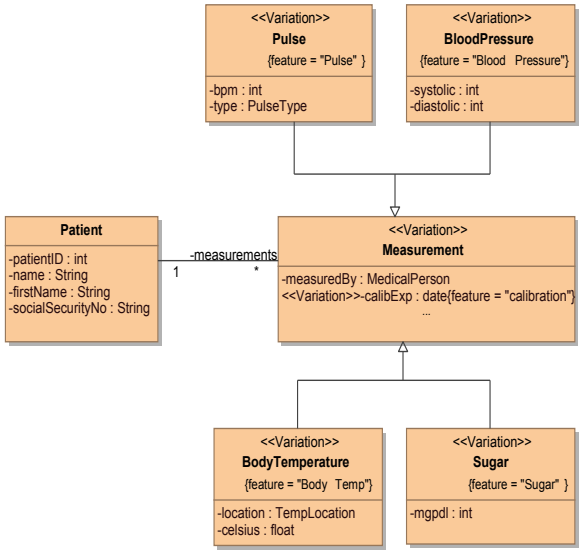


Figure 4.3. Data model with negative variability

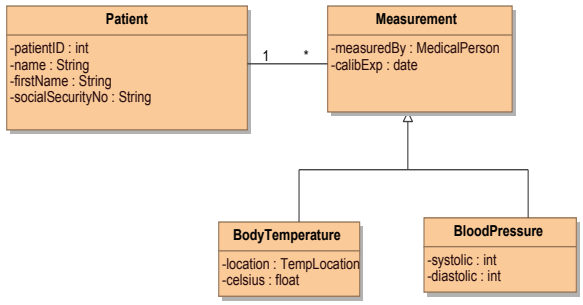


Figure 4.4. UML diagram of the example data model instance

Positive variability (Step 3)**.** Positive variable is demonstrated by the history feature, where the changes to each domain object over time should be tracked. Each domain object receives an additional member variable "history" containing previous entries and several operations. In ADMV, positive variability is realized by the stereotype <<add>> and the feature condition in brackets. Figure 4.5 shows an example.

The elements which will be added to the variation points by positive variability are composed in the class `HistoryElements`. To implement these elements via positive variability, the owner class is assigned with the stereotype <<add>> and the feature condition "History". The example reveals a scenario when positive variability is appropriate. Using negative variability to achieve the same behavior is more complex, especially the more members depend on the

cross-cutting feature: it would have to be repeated in each class that could potentially have that feature enabled and would have to be tagged with <<Variation>>. By choosing positive variability, the elements have to be modeled only once.

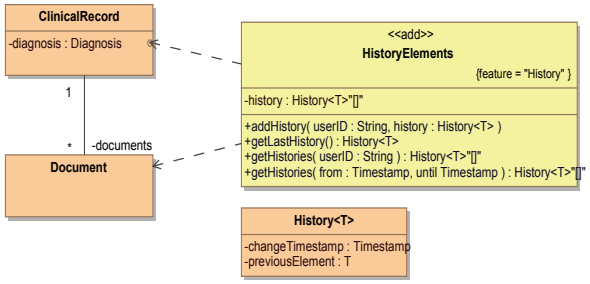


Figure 4.5. Positive variability

Structural variability (Step 3)**.** Structural variability is tagged using the sterotype <<modify>>, introducing the condition and type modification (again, in brackets: {}). For instance, see the association `records` from class `Patient` to `ClinicalRecord` in Figure 4.6.

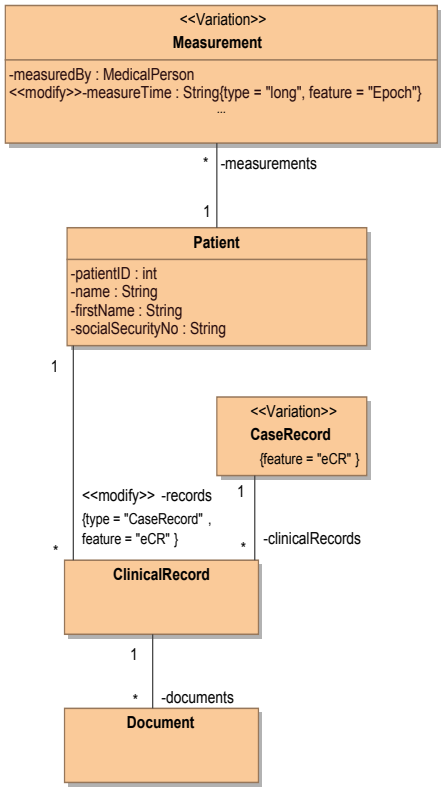


Figure 4.6. Data model with structural variability

In this example the eCR feature introduces an additional level for structuring the medical records, which is reflected by tagging the association from patient to clinical record as <<modify>> and redirecting the association to `CaseRecord` if feature "eCR" is selected. A second structural variation point is tagged to the member `measureTime` of the class `Measurement`. The date format is usually a `String`, but if "Epoch" is selected in the feature model, the date format will be a `long` (seconds since epoch).

Generated views (Step 5). The example of hierarchically differently structured patient information is defined by the structural variation point assigned to the reference `records` between `Patient` and the `ClinicalRecord` (see Figure 4.6). By default ("eCR" is not selected) records directly reference clinical records. Once the feature "eCR" is selected, the patient member `records` references `CaseRecord` which in turn references the `ClinicalRecord`. The two instances of the data model are shown for comparison in Figure 4.7.

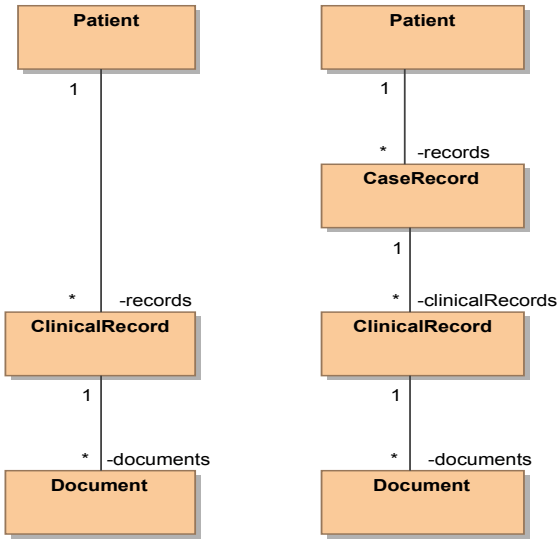


Figure 4.7. Data model results of structural variability

Generated data views for the structural variability example are shown in Listing 4.1 and different usage examples of generated data types are shown in Listing 4.2. The actual differences are written in **bold** font.

Listing 4.1

```
public interface Patient {
 List<ClinicalRecord> getRecords();
 void addRecords(ClinicalRecord value);
 void addRecords(List<ClinicalRecord>
       valueList);
    ...
}
```

```
public interface Patient {
  List<CaseRecord> getRecords();
  void addRecords(CaseRecord value);
  void addRecords(List<CaseRecord> valueList);
    ...
}
```

Listing 4.2

```
//default hierarchy
public void test1(Patient p) {
    p.getRecords().get(0).getDocuments();
}

//eCR 3-level hierarchy
public void test2(Patient p) {
   p.getRecords().get(0).getClinicalRecords()
     .get(0).getDocuments();
}
```

Consistency checks. To avoid inconsistencies in the generated artifacts, multiple checks which are defined in OCL and in the oAW Check language are executed. Listing 4.3 shows a simple oAW Check to verify the uniqueness of members. The check is applied to all attributes in the data model. If there are more than one equally named attributes within the same class, an error message informs the developer.

Listing 4.3

```
context Attribute ERROR "name not unique" :
((Class)eContainer).attributes
  .select(a|a.name == name).size == 1;
```

To verify the consistency between the feature model and the data model the following oAW check in Listing 4.4 is applied to all variation points.

Listing 4.4

```
context VariationPoint ERROR
"feature does not exist in feature model":
  getAllFeatures(featureModelUri())
  .contains( feature );
```

The function `getAllFeatures()` expects the path to the feature model as an input parameter which is resolved by `featureModelUri()`. It then returns the list of features of the feature model. Eventually the function `contains(..)` checks if the feature of the variation point in the data model is also contained in the feature model. If this evaluates to false it results in an error message.

Accessor constraints. The capability of defining runtime checks for accessors is illustrated with a security feature. The two security alternatives in the feature tree are bound to different access policies to measurements. Level 1 allows all the medical team personnel (usually, the staff in the same ward who cares for a patient) to access the measurements, Level 2 is stricter in the sense that each user may see only measurements made by themselves). Figure 4.8 shows the two examples of accessor constraints. Furthermore, accessor constraints can be associated with features as shown in Figure 4.8, again defined in brackets: {}.

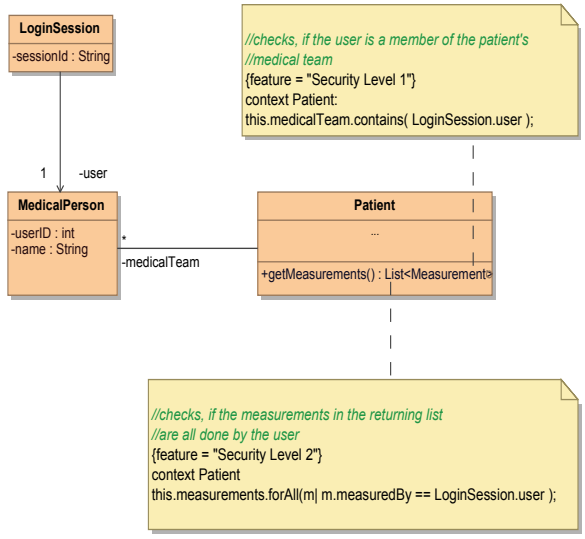


Figure 4.8. Accessor constraints

Run-time checks. Listing 4.5 shows the result of the generation process if "Security Level 1" is selected.

Listing 4.5

```
public void check1_getMeasurements()
   throws ConditionExceptionCheck1{
 if ( ! this.getMedicalTeam()
     .contains( LoginSession.getUser()) )
  throw new ConditionExceptionCheck1();
}

public List<Measurement> getMeasurements() {
 try {
  check1_getMeasurements();
 } catch (ConditionExceptionCheck1 e) {
  // resolve in an error message
 }
 return this.measurements;
}
```

The getter method calls `check1_getMeasurements()`, which checks if the logged-in user is a member of the patient's medical team. If this is not the case, then an exception will be thrown. The content of the catch block must be manually coded (Step 6) to perform further actions

such as logging, informing the user that he is not allowed to access the measurements, and returning `null`.

The authorization requirements may even be stricter. This is presented by the constraint which is associated to the feature "Security Level 2". It checks if the measurements in the returning list are all done by the logged-in user. The generated and manually added code for this check is depicted in Listing 4.6. Again the content of the catch block must be manually coded to the specific needs (Step 6), e.g., by filtering the returning list to fulfill the security rule. The manually added filtering is formatted in **bold**.

Listing 4.6

```
public void check1_getMeasurements()
   throws ConditionExceptionCheck1 {
 boolean cond = true;
 for (Measurement m : this.getMeasurements())
 {
  cond &= m.getMeasuredBy()
    == LoginSession.getUser();
 }
 if (! cond )
  throw new ConditionExceptionCheck1  ();
}

public List<Measurement> getMeasurements() {
 try {
  check1_getMeasurements();
 } catch (ConditionExceptionCheck1 e) {
  List<Measurement> filteredList
      = new ArrayList<Measurement>();
  for (Measurement m : this.getMeasurements())
  {
   if (m.getMeasuredBy()
         == LoginSession.getUser())
    filteredList.add(m);
  }
  return filteredList;
 }
 return this.measurements;
}
```

Adapters and views. The eHealth SPL contains components that operate on the clinical records independent of the context being an electronic case record infrastructure or a standard hospital. Instantiating the eCR data model would invalidate all code that uses the (simple) patient API. This illustrates the need for adapters.

The ADMV approach uses adapters to shield the actual designers and programmers from the differences in the instantiated data model. They need not be concerned about the "eCR" variation; the view – in this case – flattens the hierarchy of case records and resorts the clinical records together. Figure 4.9 shows the complete view to the left and the projected view to the right. The common elements are omitted for clarity.

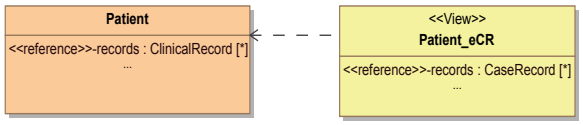


Figure 4.9. View of Patient

The ADMV Generator creates an adapter for the patient and two convert methods for each view to support bidirectional conversion (Step 5). Listing 4.7 shows a generated adapter and manually added code (Step 6) **in bold.**

Listing 4.7

```
public class Patient_eCR_Adapter
implements IPatient_eCR
{
  private IPatientView srcView;
  private IPatient_eCR adaptedView
           = (IPatient_eCR) new Patient_eCR();

  public Patient_eCR_Adapter
             (IPatientView srcView)
  {
    this.srcView = srcView;
    PatientViewConverter.convert(srcView,
                    adaptedView);
  }

  public List<CaseRecord> getRecords()
  {
    return adaptedView.getRecords();
  }
}

// there can be many convert methods
// depending on the no. of views
// the right method will be called via
// multi-method dispatching

public static void convert(Patient srcView,
   Patient_eCR targetView)
{
  // Bold code must be manually added
  CaseRecord cr = new CaseRecord();
  cr.setClinicalRecords(srcView.getRecords());
  targetView.setCaseRecords(cr);
  ...
}

public static void convert(...)
  ...
}
```

Using adapters for data integration. In case multiple devices deliver measurement data slightly differently, these must be converted to a specified core data structure. E.g., body temperature may be delivered as `value: int; scale: enum, celsius: float`, or `fahrenheit: int` from the different devices. The systems normative data structure assumes `celsius: float`, so all others need to be converted. New formats may arise at run-time too, e.g.,

when the hospital buys new devices or hospitals with different devices are merged.

The example in Figure 4.10 models a view containing a measurement type with a float representing the temperature in degrees Fahrenheit (Step 3). The resulting conversion types are generated (Step 5) and the bodies of the methods have to be added (Step 6).

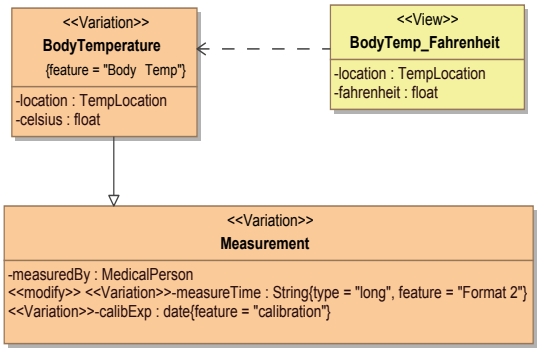


Figure 4.10. View of BodyTemperature

Listing 4.8 shows the conversion adapter for integrating measurement devices.

Listing 4.8

```
public static void convert(BodyTemp_Fahrenheit
    srcView, BodyTemperature targetView)
{
  targetView.setCelcius((
    srcView.getFahrenheit()-32)/1.8);
  …
}
```

Adapter generation. Listing 4.9 shows a simplified illustration of the Xpand-Template for the adapter generation. The input parameter for the adapter template can be any view. The French quotation marks « and » serve to distinguish between the static output and escaped control code that is interpreted. The instructions which fill the adapter template with model data are formatted here in bold.

Listing 4.9

```
«DEFINE adapterTmpl FOR View»
«FILE name + "_Adapter.java"»
public class «name»_Adapter
implements I«name»
{
  private I«entityName»View srcView;
  private I«name» adaptedView
           = (I«name») new «name»();

  public «name»_Adapter
            (I«entityName»View srcView)
  {
    this.srcView = srcView;
    «entityName»ViewConverter.convert(srcView,
                    adaptedView);
  }
  «FOREACH attributes AS a»
  public «a.type» get«a.name»()
  {
    return adaptedView.get«a.name»();
  }
  «ENDFOREACH»
  ...
«ENDDEFINE»
```

In order to convert the source view to the adapted view, the converter methods are generated (Step 5). This is done by the template in Listing 4.10. The converter template expects a complete view as an input parameter. The converter methods are generated in two steps: first all conversions from the complete view to the projected views are generated followed by all conversions in the reverse direction. When generating a convert method, it checks if the target attribute is also contained in the source view. If so, the conversion is a simple pass-through of data and can be generated automatically. Otherwise, it has to be implemented manually (Step 6).

Listing 4.10

```
«DEFINE converterTmpl FOR CompleteView»
«FILE name + "ViewConverter.java"»

public class «name + "ViewConverter"»{
«FOREACH views AS target»
  private void convert(«name
+ " src, " + target.name + " target" »){

  «FOREACH target.attributes AS attrib»
    target.set«attrib.name»(
    «IF attributes
       .select(e|e.name == attrib.name
        && e.type == attrib.type).size > 0 -»
      src.get«attrib.name»() );
    «ELSE-»
      null );
    «ENDIF-»
  «ENDFOREACH»
  }
«ENDFOREACH»
«FOREACH views AS src»
  private void convert(«src.name
  + " src, " + name + " target" »){
  ...
  }
«ENDFOREACH»
...
«ENDDEFINE»
```

5. Alternatives

This section considers various alternatives for dealing with data variability within the constraints set forth in Section 2.

UML2 package merge. The most viable alternative for data model variability is the "package merge" feature [29] introduced in UML2, and its usage for

SPLs has been evaluated [17][10]. Class extensions (e.g., additional members) can be modeled in a separate package that must have a merge association with the base package. The mapping is done on class name equality. Package merge is not suitable for the requirements described in Section 2 because it scatters the variation point over multiple packages. Thus the number of packages explodes and does not scale well with the number of features. Package merge cannot model negative or structural variability that is needed for requirement 2.

To compare the ability of both approaches, the number of classes to model was counted. The Measurement class and its subtypes result in five classes, four of which are tagged to be variable, two members are also tagged.

Using package merge, a core model containing only the base class Measurement without the members `calibExp` and `measuredTime` was used. For each subtype, a package was created since each of the subtypes is selectable separately. Within the package, the base class is repeated to add the child class and the association between them. For the optional calibration management, a package is added, repeating the base class with the member `calibExp`. For the structural variability of the `measuredTime`, two packages are needed, one repeating the base class with the `long` type and one with the `String` type. This requires up to seven additional packages and 12 classes. Additionally, data model comprehension becomes difficult since the information is spread across many packages (see Figure 5.1).

A general comparison of the effort involved in the two approaches is shown in Table 5.1, where:

F = number of features influencing the data model

$V(f)$ = number of variation elements for a single feature f

$C(f)$ = Number of classes created or modified by feature f (might be less then $V(f)$ in case a feature controls more than one member of a class)

Table 5.1. UML Package Merge vs. ADMV for packages, attributes, and classes

Approach	Packages	Attributes created or modified	Repeated Classes
UML package merge	F	$\sum_{f=1}^{F} V(f)$	$\sum_{f=1}^{F} C(f)$
ADMV	0	$\sum_{f=1}^{F} V(f)$	0

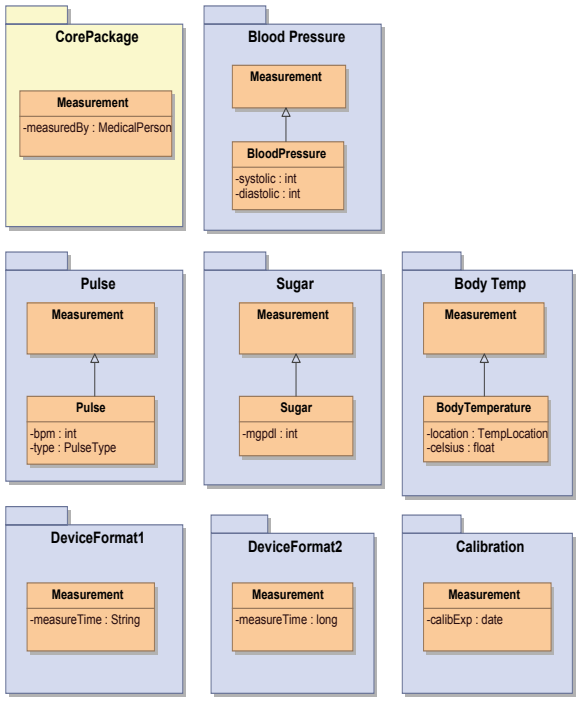


Figure 5.1. UML Package Merge

Optional members. Negative variability could be modeled by using a full-blown data model for each instance and returning "null" in case a non-selected member or association is requested. Alternatively, hashmaps could be used to carry (single-valued) optional members. Shortcomings of this approach include:

- Members cannot be declared to be not null, in case the feature is selected and null is an inappropriate value (especially if the data model is persisted in databases).
- The development of all components could accidentally use members that are not necessarily selected. Auto-completion and compile-time checks are not possible.
- Using hashmaps gives developers no indication about available members.
- Structural variability is not possible.

Explicit dependencies. Each extension to the data model could be presented by a separate data component and explicitly used by a functional component (see Figure 5.2). The data components retrieve the necessary elements to form their view on a domain data. Communicating with other components introduces the obligation for the receiving component to retrieve their view of the data again.

Drawbacks include the numerous calls for database retrievals per component due to a lack of sharing, as

well as interpretation difficulties when components transmit or receive data via references or value objects. If transactions are considered or the services are remote, this solution is infeasible.

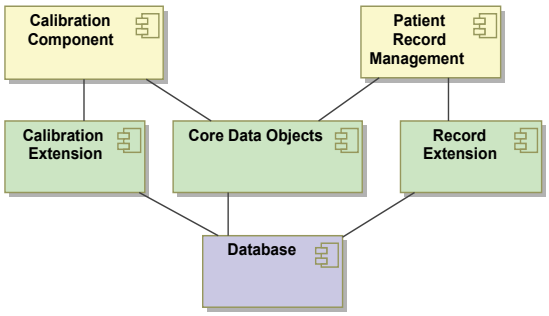


Figure 5.2. Data model with explicit dependencies

Layering. Similar to the Decorator design pattern [6], components are grouped in layers that correspond to the level of enrichment of the data (see Figure 5.3). Per layer, one definition of each data element exists. On the lowest level this will be a core element, on the next level a slightly enriched element (some more attributes or associations) that can even be extended on higher levels. If a component of a higher layer needs data, it asks the persistence component of that layer to retrieve it. The persistence component routes this request down the layers and extends ("enriches") the data, converting the data into its own layer data model.

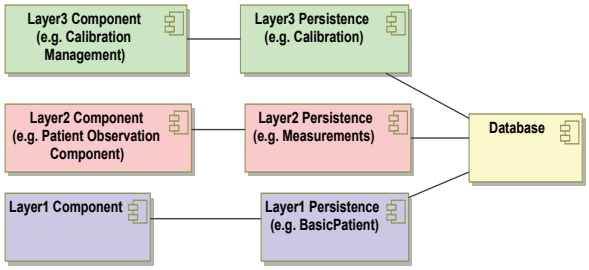


Figure 5.3. Data model with layered extensions

Multiple calls have to be executed to provide the Layer 3 component with the data and there is insufficient support for transaction handling.

In summary, the aforementioned alternatives have the disadvantage that the data model is not presented consistently and that knowledge is spread over at least some layers or even individual components. This assumes that a strict layering is even possible. The ADMV does not create separate information in separate packages nor does it need to repeat classes as UML package merge does. Feature-dependent classes are defined once in both approaches (see the `Measurement` children).

6. Evaluation

For an evaluation of the ADMV, an appraisal of its support for desirable qualities is considered. Additionally, any practical limitations of the approach with regard to performance and scalability with current implementations are also assessed.

6.1 Quality properties

Consistency and correctness. Correctness is supported via constraint checks and the generation of adapters and projected data views appropriate for a component in its current version. Via the validation of the data model at usage time via OCL constraints (e.g., MagicDraw Active validation), various modeling errors can be detected sooner and thus avoided in later phases. Consistency checks can assure the consistency of the models, e.g., between the feature model and data model. Support for the correctness of the generated artifacts is thus enhanced.

Comprehension. ADMV reduces the number of classes and locations where (redundant) information is stored, which furthers comprehension. Code generation is based on a metamodel specialized for modeling data variability. Code generation templates can thus be more simply created compared to UML metamodel generative approaches such as OMG's Model-Driven Architecture (MDA).

Maintainability. Maintenance and evolvability are supported by both shielding component developers from changes via adapters as well as the application of constraints throughout development. By programming templates against a common ADMV metamodel, an unlimited number of future templates and template changes support any necessary extensibility.

Usability. Usability is fostered by the integration of ADMV in standard modeling (FM and UML) and with tool frameworks that support customization (e.g., oAW). The usage of constraint languages at the appropriate levels also furthers usability.

Efficiency. The enhanced support for code generation techniques has the potential to improve efficiency for larger SPLs. Runtime efficiencies are also achievable since variation decisions are typically made at generation time. The reduction in the number of classes required to deal with variability also promotes efficiency.

Portability. Modeling variability with UML-based stereotypes, coupled with the ADMV metamodel as a

basis for generation, supports the portability and exchangeability of MDSD implementations for modeling, model-to-model transformations, and artifact generation.

Data integration and interoperability. These qualities are supported via the adapters and projected component views that support independent conversions. The complete view also supports interoperability across the SPL and product instance life cycle.

Reusability. Component reuse is supported since no direct tight coupling to other components via data elements occurs. Enhanced comprehension enhances reusability opportunities. Templates reuse common code.

Testability. Constraints can be readily defined via very capable languages such as OCL and oAW Check, which supports the testability of the models. The reduced number of classes also simplifies component testing, since knowledge of other existing components in the individual product instances is not required.

Traceability. The modeling of variability and data in a central model makes the effects of the variability more traceable. By using UML tools with stereotypes and tagged-value-based search capabilities (as in MagicDraw), the traceability of variation points and features is improved. Certain variation points can be localized by simple string searches.

6.2 Performance and scalability

Due to the use of code generation techniques, the impact of the variations and the use of the adapters at runtime is relatively inconsequential. View conversion of data where necessary, e.g., from one format to another, is currently a manual programming task and thus the runtime impact is dependent on the conversion complexity. However, due to the large set of possible permutations and the reliance on MDD, variation scalability measurements were made to determine the impacts of the variations for development time usage of the ADMV.

The measurements were performed on an AMD Athlon XP 2400+ (2GHz) PC with 3GB RAM running Microsoft Windows XP Pro SP2, Java JDK 1.6, Eclipse 3.3, openArchitectureWare 4.2, and the Eclipse Modeling Framework 2.3. All measurements were performed 3 times and the averages presented.

For the first set of measurements, the transformation time using oAW from an XMI Data Model file containing variations to a Data Model Instance XMI (all variation points applied based on features) was measured as shown in Table 6.1 and Figure 6.1. A nearly linear correlation between a change in the number of variation points and the generation time was measured as the number of features was held constant, and an increase in the number of features also showed a nearly linear increase in the generation time. This result is explained by the iterations in the generator code implementation for each variation point and for each feature. Varying the number of Boolean conjunctions up to 20 for a variation point made no perceptible difference due to other inherent overheads.

Table 6.1. Data model instance transformation time (ms) for features and variation points

Number of variation points	Total number of features		
	300	600	900
50	2771	5281	9141
100	4429	9696	17781
150	6416	14219	26078

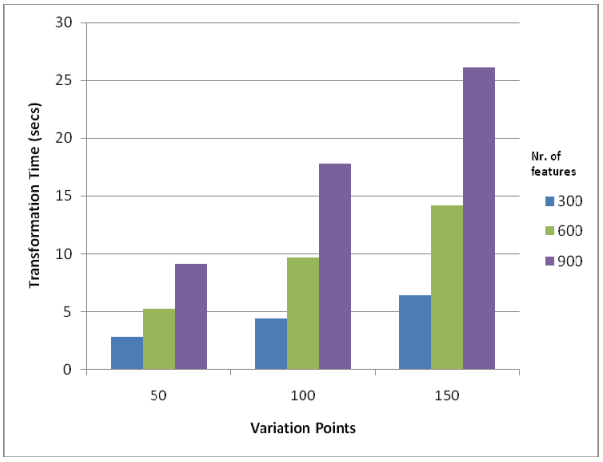


Figure 6.1. Data model instance transformation time vs. variation points and features

A second set of measurements concerned the generation of adapters. Each of the different variability types was tested and, as expected, no noticeable difference in generation time occurred based on the negative, positive, or structural variability types. In the ADMV, each adapter for an entity can support multiple projected views. The Lines of Code (LOC) generated in support of the conversion betweens views increased in the same percentage to the number of views, as expected due to the 2n relation resulting from the complete view basis for all conversions. The maintainability of the conversions is thereby supported. The generation time required for adapters with multiple views is shown in Figure 6.2, showing a nearly linear increase as the number of adapters or views increase. The generation time for this scale

appears reasonable for development usage in current SPLs.

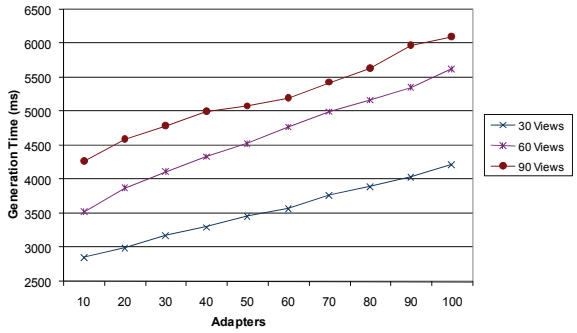


Figure 6.2. Generation time vs. number of adapters

In summary, the development-time variation scalability and performance of the ADMV with current tooling for industrial use is shown to be practicable.

7. Related work

Other approaches for SPL variability in data models include the conceptual framework SPLIT [3], where additional UML stereotypes, e.g., <<variabilityMechanism>> and <<variationPoint>>, are used for specifying variable elements. SPLIT does not, however, integrate an abstract feature view as does the ADMV, and each variation point and the corresponding variants requires a separate class which can cause issues in lucidity for large SPLs.

Clauß presents in [24] a generic modeling approach which uses additional stereotypes to express variability. The Stereotype <<optional>> is used for optional variants which do not stand in a relationship with other variants (variation point with one variant). Variation points which group multiple variants together are tagged with the stereotype <<variationPoint>> and the associated variants with <<variant>>. Furthermore, the variation points and variants can be assigned with tagged values to define certain properties. Some of these properties are the binding time of variants, the multiplicity of associable variants, and the condition of binding. However, this approach doesn't offer a concept to handle data independently from the corresponding product instance, nor does it address the derivation of product line instances.

In [7], PLUS (Product Line UML-Based Software Engineering) extends UML to model variability and commonality using stereotypes and primarily subclassing. While entities are mentioned, the wrappers described are intended for database access and do not support all variation types and multiple view and data versions for components as addressed in the ADMV. The extension of PLUS with the ADMV would provide a more comprehensive solution for SPL UML techniques.

In MDD-AO-PLE [15][16][18] and similar related aspect-oriented SPLE work, the application of techniques to SPLs are investigated for addressing cross-cutting variability. While this work has not specifically addressed the difficulties described in this paper for data models, the combination of these techniques with ADMV could be synergistic, e.g., to address positive variability or for common data view format conversions in adapters.

The following comparison matrix shows a assessment of related SPLE approaches in regard to a selection of requirements.

Table 7.1. Comparison matrix

	SPLIT	PLUS	MDD-AO-PLE	UML ext. [24]	ADMV
requirement analysis	+++	+++	++	+	++
FM[1] integration		✔	✔	✔	✔
positive variability	✔	✔	✔	✔	✔
negative variability				✔	✔
structural variability	✔	✔		✔	✔
UML2	✔	✔		✔	✔
data conversion[2]					✔
checks (modeling)	✔	✔	✔	✔	✔
checks (config.)[3]			✔	✔	✔
checks (generator)			✔	✔	✔
checks (runtime)					✔
product instantiation[4]	+++	+	+++	+	+
code generation			✔		✔
trace variability[5]	✔	✔	✔	✔	✔

([1]) FM = feature model.

([2]) Ability to convert data to different formats.

([3]) Checks at configuration time.

([4]) The process of creating a specific software product using a software product line is referred to as product instantiation [25].

([5]) Ability to trace variability between solution space and problem space.

Work with regard to SPL component evolution support includes [5], where a multi-team decentralized SPL variability modeling approach is described, supporting the merging of model fragments. However, it does not address versioning of entities and component usage and lacks UML support. [22] addresses multi-context component reusability using UML extensions views (functional, static, and dynamic), but does not consider data modeling, constraints, or code generation issues.

Work on model-based data integration, mapping, and transformation in the eHealth domain includes [21] and the AutoMed project [20]. To our knowledge the usage of such an approach for an eHealth SPL for modeling data variability has not been explored.

8. Conclusion and future work

Given the inadequate integration and specific support for MDSD data modeling and variability in current SPL approaches and research, the ADMV contributes a UML standards-based method for data modeling that can be utilized by common MDSD tooling, is integrated with feature modeling, and supports desirable software qualities during SPL development. UML diagrams are augmented with variability information including constraints, from which artifacts for particular configurations can be generated automatically. The approach for adapter generation supports SPL data integration with the potentially multifarious external systems and devices, which may represent the same kind of information in different formats

An eHealth case study that motivated the work was used to illustrate the application of the ADMV to a SPL. Scalability of the ADMV with regard to features and variation points is linear and likely to be sufficient for typical current SPL development. The unification of concepts and mechanisms in ADMV promote support for desirable SPLE qualities, including consistency, correctness, comprehension, maintainability, usability, efficiency, portability, integration, interoperability, reusability, testability, and traceability. These and other benefits can be realized for SPLs in conjunction with the ADMV.

Future work includes the addition of a conversion language for somewhat complex conversions in adapters (e.g., concatenation and regex-split). Support of dynamic runtime variation including adaptation and binding of component views with database migration is another area to be investigated. Additionally, optimization for object tree transfers and greater automatic adapter data conversion code generation are promising.

9. References

[1] Bartholdt, J., Oberhauser, R., and Andreas Rytina, "An Approach to Addressing Entity Model Variability within Software Product Lines". In *Proceedings of the Third International Conference on Software Engineering Advances (ICSEA 2008)*, IEEE Computer Society Press, 2008.

[2] Bayer, J., Flege, O., Knauber, P., Laqua, R., Muthig, D., Schmid, K., Widen, T., and DeBaud, J., "PuLSE: a methodology to develop software product lines," In *Proceedings of the 1999 Symposium on Software Reusability* (SSR '99). ACM, pp. 122-131.

[3] Coriat, M., Jourdan, J., and Boisbourdin, F., "The SPLIT method: building product lines for software-intensive systems," In *Proceedings of the First Conference on Software Product Lines: Experience and Research Directions* (Denver, Colorado, United States). P. Donohoe, Ed. Kluwer Academic Publishers, Norwell, MA, 2000, pp. 147-166.

[4] Czarnecki, K. and Eisenecker, U.W., *Generative Programming: Methods, Techniques, and Applications.* Addison–Wesley, May 2000, ISBN 0201309777.

[5] Dhungana, D., Neumayer, T., Gruenbacher, P., and Rabiser, R., "Supporting the Evolution of Product Line Architectures with Variability Model Fragments," In *Proceedings of the Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA 2008) (February 18 - 21, 2008)*. WICSA. IEEE Computer Society, Washington, DC, 2008, pp. 327-330.

[6] Gamma, E., Helm, R., Johnson, R., Vlissides, J., *Design patterns: elements of reusable object-oriented software*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 1995, ISBN 0-201-63361-2.

[7] Gomaa, H., *Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures,* Addison-Wesley, 2005, ISBN 0201775956.

[8] Griss, M. L., Favaro, J., and Alessandro, M. d. "Integrating Feature Modeling with the RSEB," In *Proceedings of the 5th international Conference on Software Reuse (June 02 - 05, 1998)*. ICSR. IEEE Computer Society, Washington, DC, 1998, p. 76-85.

[9] Kang, K. C.; Cohen, S. G.; Hess, J. A.; Novak, W. E. & Peterson, A. S. "Feature-Oriented Domain Analysis (FODA) Feasibility Study," Carnegie-Mellon University Software Engineering Institute, 1990.

[10] Laguna, M. A., González-Baixauli, B., and Marqués, J. M., "Seamless development of software product lines," In *Proceedings of the 6th international Conference on*

Generative Programming and Component Engineering (GPCE 2007). ACM, New York, NY, pp. 85-94.

[11] Linden, F.J. v.d., Schmid, K., and Rommes, E., *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*, Springer, Berlin, ISBN 3540714367, 2007.

[12] Voelter, M., Haase, A., Kolb, B., and Efftinge, S., "Introduction to openArchitectureWare 4.1.2," In *Proceedings of the Model-Driven Development Tool Implementers Forum* (MDD-TIF07), TOOLS EUROPE 2007.

[13] Batini, C., Lenzerini, M., and Navathe, S. B., "A comparative analysis of methodologies for database schema integration," ACM Computing Surveys (CSUR), Vol. 18, Issue 4, (Dec. 1986), 323-364.

[14] Pohl, K., Böckle, G., Linden, F.J. v.d., *Software Product Line Engineering: Foundations, Principles and Techniques*, Springer-Verlag, 2005, ISBN 3540243720.

[15] Voelter, M. and Groher, I., "Handling Variability in Model Transformations and Generators", in *Proceedings of the 7th OOPSLA Workshop on Domain-Specific Modeling* (DSM'07), Sprinkle, J., Gray, J., Rossi, M., Tolvanen, J.-P., (eds.), Computer Science and Information System Reports, Technical Reports, TR-38, University of Jyväskylä, Finland 2007, ISBN 978-951-39-2915-2.

[16] Voelter, M. and Groher, I., "Product Line Implementation using Aspect-Oriented and Model-Driven Software Development," In *Proceedings of the 11th international Software Product Line Conference* (September 10 - 14, 2007). International Conference on Software Product Line. IEEE Computer Society, Washington, DC, 2007, pp. 233-242.

[17] Laguna, M. A., González-Baixauli, B., and Marqués, J. M., "Seamless development of software product lines," In *Proceedings of the 6th international Conference on Generative Programming and Component Engineering* (Salzburg, Austria, October 01 - 03, 2007). GPCE '07. ACM, New York, NY, 2007, pp. 85-94.

[18] Groher, I., "Aspect-Oriented Feature Definitions in Model-Driven Product Line Engineering", Dissertation, Johannes Kepler Universität, Linz, April 2008.

[19] Boehm, O., "eCR Application Architecture v1.2 Services and Interfaces", Fraunhofer Institute for Software and Systems Engineering (ISST), www.fallakte.de 2008

[20] Smith, A. and Mcbrien, P., "A Generic Data Level Implementation of ModelGen," In *Proceedings of the 25th British National Conference on Databases: Sharing Data, information and Knowledge* (Cardiff, Wales, UK, July 07 - 10, 2008). A. Gray, K. Jeffery, and J. Shao, Eds. Lecture Notes In Computer Science, vol. 5071. Springer-Verlag, Berlin, Heidelberg, 2008, pp. 63-74.

[21] Ying, B., Rong, Z., and Xiao, J., "A Data Integration Approach to E-Healthcare System". In *Proceedings of the 1st International Conference on Bioinformatics and Biomedical Engineering, 2007 (ICBBE 2007)*, pp. 1129 – 1132.

[22] Saidi, R., Front, A., Rieu, D., Fredj, M., Mouline, S., "From a Business Component to a Functional Component using a Multi-View Variability Modelling," In *Proceedings of the International Workshop on Model Driven Information Systems Engineering: Enterprise, User and System Models (MoDISE-EUS'08)* held in conjunction with the CAiSE'08 Conference, Montpellier, France, June 16-17, 2008, ISSN 1613-0073, pp. 34-45.

[23] Antkiewicz, M. and Czarnecki, K., "FeaturePlugin: feature modeling plug-in for Eclipse," In *Proceedings of the 2004 OOPSLA Workshop on Eclipse Technology Exchange* (Vancouver, British Columbia, Canada, October 24 - 24, 2004), eclipse '04, ACM, 2004, pp. 67-72.

[24] Clauss M., "Generic modeling using UML extensions for variability", In *Proceedings of the Workshop on Domain Specific Visual Languages*, OOPSLA 2001, Jyväskylä University Printing House, Jyväskylä, Finland, 2001, ISBN 951-39-1056-3, pp. 11-18.

[25] Van Gurp, J., Bosch, J., and Svahnberg, M. "On the Notion of Variability in Software Product Lines," In *Proceedings of the Working IEEE/IFIP Conference on Software Architecture (August 28 - 31, 2001)*. WICSA. IEEE Computer Society, Washington, DC, 2001, pp. 45-54.

[26] Eclipse Modeling Framework Project, http://www.eclipse.org/modeling/emf/ May 14, 2009.

[27] Vlissides, J., "Generation Gap," C++ Report Volume 8, Number 10, November / December, 1996, pp. 12-18.

[28] Generic Eclipse Modeling System (GEMS), http://www.eclipse.org/gmt/gems/ May 14, 2009.

[29] OMG, "UML 2.1.2 Superstructure Specification", OMG doc# formal/07-11-02, 2007.

[30] OMG, "Meta Object Facility(MOF) 2.0 XMI Mapping Specification, v2.1.1", OMG doc# formal/07-12-01.

From Supervised to Reinforcement Learning: a Kernel-based Bayesian Filtering Framework

Matthieu GEIST[1,2,3], Olivier PIETQUIN[1] and Gabriel FRICOUT[2]
[1]IMS Research Group, Supélec, Metz, France
[2]MC Cluster, ArcelorMittal Research, Maizières-lès-Metz, France
[3]CORIDA Project-Team, INRIA Nancy - Grand Est, France
{matthieu.geist,olivier.pietquin}@supelec.fr

***Abstract*—In a large number of applications, engineers have to estimate a function linked to the state of a dynamic system. To do so, a sequence of samples drawn from this unknown function is observed while the system is transiting from state to state and the problem is to generalize these observations to unvisited states. Several solutions can be envisioned among which regressing a family of parameterized functions so as to make it fit at best to the observed samples. This is the first problem addressed with the proposed kernel-based Bayesian filtering approach, which also allows quantifying uncertainty reduction occurring when acquiring more samples. Classical methods cannot handle the case where actual samples are not directly observable but only a non linear mapping of them is available, which happens when a special sensor has to be used or when solving the Bellman equation in order to control the system. However the approach proposed in this paper can be extended to this tricky case. Moreover, an application of this indirect function approximation scheme to reinforcement learning is presented. A set of experiments is also proposed in order to demonstrate the efficiency of this kernel-based Bayesian approach.**

***Index Terms*—supervised learning; reinforcement learning; Bayesian filtering; kernel methods**

I. Introduction

In a large number of applications, engineers have to estimate values of an unknown function given some observed samples. For example, in order to have a map of wifi (wireless fidelity) coverage in a building, one solution would be to simulate the wave propagation in the building according to Maxwell equations, which would be intractable in practice. An other solution is to measure the electromagnetic field magnitude in some specific locations, and to interpolate between theses observations in order to build a field map which covers the whole building. This task is referred to as function approximation or as generalization. One way to solve the problem is to regress a family of parameterized functions so as to make it fit at best the observed samples. Lots of existing regression methods can be found in the literature for a wide range of function families. Artificial Neural Networks (ANN) [2] or kernel machines [3], [4] are popular methods. Yet, usually batch methods are used (gradient descent for ANN or Support Vector Regression for kernel machines); that is all the observed samples have to be known before regression is done. A new observed sample requires running again the regression algorithm using every sample.

Online regression describes a set of methods able to incrementally improve the regression results as new samples are observed by recursively updating previously computed parameters. There exists online regression algorithms using ANN or kernel machines, yet the uncertainty reduction occurring when acquiring more samples (thus more information) is usually not quantified, as well as with the batch methods. Bayesian methods are such recursive techniques able to quantify uncertainty about the computed parameters. They have already been applied to ANN [5], [6] and, to some extent, to kernel machines [7], [8]. In this paper is proposed a method based on the Bayesian filtering framework [9] for recursively regressing a nonlinear function from noisy samples. In this framework a hidden state (here the regression parameter vector) is recursively estimated from observations (here the samples), while maintaining a probability distribution over parameters (uncertainty estimation).

Several problems are not usually handled by standard techniques. For instance, actual samples are sometimes not directly observable but only a non linear mapping of them is available. This is the case when a special sensor has to be used (*e.g.*, measuring a temperature using a spectrometer or a thermocouple). This is also the case when solving the Bellman equation in a Markovian decision process with unknown deterministic transitions [10]. This is important for (asynchronous and online) dynamic programming, and more generally for control theory. The proposed approach is extended to online regression of nonlinear mapping of observations. First a quite general formulation of the problem is described in order to appeal a broader audience. Indeed the technique introduced below handles well nonlinearities in a derivative free way and it can be useful in other fields. Nevertheless, an application of this framework to reinforcement learning [11] is also described. The general outline of the proposed method is as follows.

The parametric function approximation problem as well as its extension to nonlinear mapped observations case mainly breaks down in two parts. First, a representation for the approximated function must be chosen. For example, it can be an ANN. Notice that this also involves to choose a specific structure, *e.g.*, number of hidden layers, number of neurons, synaptic connections, *etc*. A kernel representation is chosen in this paper, because of its expressiveness given by the Mercer theorem [4]. Moreover a dictionary method [12] allows quasi-

automatizing the choice of the associated structure (that is number and positions of kernel basis). Second, an algorithm to learn the parameters is necessary. For this, the regression problem is cast in a Bayesian tracking problem [9]. As it will be shown, it allows handling well nonlinearities, uncertainty and even non-stationarity.

The next section presents some necessary background: the dictionary method and Bayesian filtering. The following sections describe the proposed algorithm for function approximation [1], its extension to regression from nonlinear mapping of observations [13] and the application of this general algorithm to reinforcement learning [14]. All these algorithms are experimented and compared to the state-of-the-art, and the last section concludes.

II. Background

Before introducing the proposed approach, some backbone methods are presented. The first one is a dictionary method [12] based on mathematical signification of kernels and basic linear algebra which allows automatizing the choice of the structure (number and position of kernels). The second one, Bayesian filtering and more precisely Sigma Point Kalman filtering, is used as the learning part of the proposed algorithms. First, kernel-based regression is briefly introduced.

A. Kernel-based Regression

A kernel-based regression is used, namely the approximation is of the form $\hat{f}_\theta(x) = \sum_{i=1}^{p} \alpha_i K(x, x_i)$ where x belongs to a compact set $\mathcal{X}$ of $\mathbb{R}^n$ (all the work is done in $\mathcal{X}$) and K is a kernel, that is a continuous, symmetric and positive semi-definite function. The parameter vector θ contains the weights $(\alpha_i)_i$, and possibly the centers $(x_i)_i$ and some parameters of the kernel (*e.g.*, the variance for Gaussian kernels). These methods rely on the Mercer theorem [4] which states that each kernel is a dot product in a higher dimensional space. More precisely, for each kernel K, there exists a mapping $\varphi : \mathcal{X} \to \mathcal{F}$ ($\mathcal{F}$ being called the feature space) such that $\forall x, y \in \mathcal{X}$, $K(x, y) = \langle \varphi(x), \varphi(y) \rangle$. Thus, any linear regression algorithm which only uses dot products can be cast by this *kernel trick* into a nonlinear one by implicitly mapping the original space $\mathcal{X}$ to a higher dimensional one. Many approaches to kernel regression can be found in the literature, the most classical being the Support Vector Machines (SVM) framework [4]. There are fewer Bayesian approaches, nonetheless the reader can refer to [7] or [8] for interesting examples. To our knowledge, none of them is designed to handle the second regression problem described in this paper (when observations are nonlinearly mapped).

B. Dictionary

A first problem is to choose the number p of kernel functions required for the regression task and the prior kernel centers. A variety of methods can be contemplated, the simplest one being to choose equally spaced kernel fonctions. However the method described below rests on the mathematical signification of kernels and basic algebra, and is thus well motivated. By observing that although the feature space $\mathcal{F}$ is a (very) higher dimensional space, $\varphi(\mathcal{X})$ can be a quite smaller embedding, the objective is to find a set of p points in $\mathcal{X}$ such that

$$\varphi(\mathcal{X}) \simeq \mathrm{Span}\left\{\varphi(\tilde{x}_1), \ldots, \varphi(\tilde{x}_p)\right\} \tag{1}$$

This method is iterative. Suppose that samples $x_1, x_2, \ldots$ are sequentially generated. At time k, a set

$$\mathcal{D}_{k-1} = (\tilde{x}_j)_{j=1}^{m_{k-1}} \subset (x_j)_{j=1}^{k-1} \tag{2}$$

of m_{k-1} elements is available where by construction feature vectors $\varphi(\tilde{x}_j)$ are approximately linearly independent in $\mathcal{F}$. A sample x_k is then uniformly sampled from $\mathcal{X}$, and is added to the dictionary if $\varphi(x_k)$ is linearly independent on $\mathcal{D}_{k-1}$. To test this, weights $a = (a_1, \ldots, a_{m_{k-1}})^T$ have to be computed so as to verify

$$\delta_k = \min_{a \in \mathbb{R}^{m_{k-1}}} \left\| \sum_{j=1}^{m_{k-1}} a_j \varphi(\tilde{x}_j) - \varphi(x_k) \right\|^2 \tag{3}$$

Formally, if $\delta_k = 0$ then the feature vectors are linearly dependent, otherwise not. Practically an approximate dependence is allowed, and δ_k is compared to a predefined threshold ν determining the quality of the approximation (and consequently the sparsity of the dictionary). Thus the feature vectors will be considered as approximately linearly dependent if $\delta_k \leq \nu$.

By using the kernel trick and the bilinearity of dot products, equation (3) can be rewritten as

$$\delta_k = \min_{a \in \mathbb{R}^{m_{k-1}}} \left\{ a^T \tilde{K}_{k-1} a - 2 a^T \tilde{k}_{k-1}(x_t) + K(x_k, x_k) \right\} \tag{4}$$

where

$$\left(\tilde{K}_{k-1}\right)_{i,j} = K(\tilde{x}_i, \tilde{x}_j) \tag{5}$$

is a $m_{k-1} \times m_{k-1}$ matrix and

$$\left(\tilde{k}_{k-1}(x)\right)_i = K(x, \tilde{x}_i) \tag{6}$$

is a $m_{k-1} \times 1$ vector. If $\delta_k > \nu$, $x_k = \tilde{x}_{m_k}$ is added to the dictionary, otherwise not. Equation (4) admits the following analytical solution

$$\begin{cases} a_k = \tilde{K}_{k-1}^{-1} \tilde{k}_{k-1}(x_k) \\ \delta_k = K(x_k, x_k) - \tilde{k}_{k-1}(x_k)^T a_k \end{cases} \tag{7}$$

Notice that the matrix $\tilde{K}_k^{-1}$ can be computed efficiently. If $\delta_k \leq \nu$ no point is added to the dictionary, and thus $\tilde{K}_k^{-1} = \tilde{K}_{k-1}^{-1}$. If x_k is added to the dictionary, one can write the matrix $\tilde{K}_k$ by blocs:

$$\tilde{K}_k = \begin{pmatrix} \tilde{K}_{k-1} & \tilde{k}_{k-1}(x_k) \\ \tilde{k}_{k-1}(x_k)^T & K(x_k, x_k) \end{pmatrix} \tag{8}$$

By using the partitioned matrix inversion formula, its inverse is incrementally computed:

$$\tilde{K}_k^{-1} = \frac{1}{\delta_k} \begin{pmatrix} \delta_k \tilde{K}_{k-1}^{-1} + a_k a_k^T & -a_k \\ -a_k^T & 1 \end{pmatrix} \tag{9}$$

where a_k and δ_k are the given analytical solution to problem (4). This bounds the computational cost for the k^{th} sample by $O(m_k^2)$. Thus this approach allows computing sequentially and incrementally an approximate basis of $\varphi(\mathcal{X})$. The dictionary method is briefly sketched in Algorithm 1, nevertheless see [12] for more details and theoretical analysis of the properties of this approach.

Algorithm 1: Dictionary computation

inputs : a set of N samples randomly selected from $\mathcal{X}$, sparsification parameter ν

outputs: a dictionary $\mathcal{D}$

Initialization;

$\mathcal{D}_1 = \{x_1\}$;

Dictionary computation;

for $k = 1, 2, \ldots N$ **do**

Observe sample x_k;

Compute approximate dependence:

$$\delta_k = \min_{a \in \mathbb{R}^{m_{k-1}}} \left\| \sum_{j=1}^{m_{k-1}} a_j \varphi(\tilde{x}_j) - \varphi(x_k) \right\|^2$$

if $\delta_k > \nu$ **then**

Add x_k to the dictionary: $\mathcal{D}_k = \mathcal{D}_{k-1} \cup \{x_k\}$

else

Let the dictionary unchanged: $\mathcal{D}_k = \mathcal{D}_{k-1}$

Thus, by choosing a prior on the kernel to be used, and by applying this algorithm to a set of points $(x_1, \ldots, x_N)$ randomly sampled from $\mathcal{X}$, a sparse set of good candidates to the kernel regression problem is obtained. This method is theorically well founded, easy to implement, computationally efficient and it does not depend on kernels nor space topology. Notice that, despite the fact that this algorithm is naturally online, this dictionary cannot be built (straightforwardly) while estimating the parameters, since the hyper-parameters of the chosen kernels (such as mean and deviation for Gaussian kernels) will be parameterized as well (which leads to a nonlinear parameterization). If the samples used for regression are known beforehand, they can be used to construct the dictionary. However, for online regression, samples are generally not known beforehand. Knowing bounds on $\mathcal{X}$ is sufficient to compute a dictionary.

C. Bayesian Filtering

Bayesian filtering was originally designed to track the state of a stochastic dynamic system from observations (*e.g.*, tracking the position of a plane from radar measures). When used as a learner for function approximation, the parameter vector is the hidden state to be tracked. As it will be shown below, this parameter vector is modeled as a random variable, whereas it is generally deterministic. However, this allows handling non-stationary regression problems, and even if stationary, it helps avoiding local minima. Indeed, the process noise (to be defined below) then plays a role quite similar to simulated annealing. A comprehensive survey of Bayesian filtering is given in [9].

1) Paradigm: The problem of Bayesian filtering can be expressed in its state-space formulation; suppose that the dynamic of a system and the associated generated observations are driven by the following equations:

$$\begin{cases} s_{k+1} = f_k(s_k) + v_k \\ y_k = g_k(s_k) + n_k \end{cases} \tag{10}$$

The objective is to sequentially infer the hidden state s_k given the observations $y_1, \ldots, y_k = y_{1:k}$. The state evolution is driven by the possibly nonlinear mapping f_k and the process noise v_k (centered and of variance P_{v_k}). The observation y_k is a function of the state s_k, corrupted by an observation noise n_k (centered and of variance P_{n_k}). To do so, the posterior density (of state over past observations) is recursively updated as new observations arrive by making use of the Bayes rule and of the dynamic state-space model of the system (10).

Such a Bayesian filtering approach can be decomposed in two steps, which crudely consists in predicting the new observation generated by the system given the current approximated model, and then correcting this model according to the accuracy of this prediction, given the new observation. The first stage is the prediction step. It consists in computing the following distribution:

$$p(S_k|Y_{1:k-1}) = \int_{\mathcal{S}} p(S_k|S_{k-1}) p(S_{k-1}|Y_{1:k-1}) dS_{k-1} \tag{11}$$

It is the prior distribution of current state conditioned on past observations up to time $k-1$. It is a projection forward in time of the posterior at time $k-1$, $p(S_{k-1}|Y_{1:k-1})$ by using the process model represented by $p(S_k|S_{k-1})$ which depends on f_k. For example, if the evolution function is linear and the noise is Gaussian, the distribution of $S_k|S_{k-1}$ is Gaussian of mean $f_{k-1}(S_{k-1})$ and of variance $P_{v_{k-1}}$:

$$S_k|S_{k-1} \sim \mathcal{N}\left(f_{k-1}(S_{k-1}), P_{v_{k-1}}\right) \tag{12}$$

Second, the noisy measurement is incorporated using the observation likelihood and is combined with the prior to update the posterior. This is the correction step:

$$p(S_k|Y_{1:k}) = \frac{p(Y_k|S_k) p(S_k|Y_{1:k-1})}{\int_{\mathcal{S}} p(Y_k|S_k) p(S_k|Y_{1:k-1}) dS_k} \tag{13}$$

The likelihood $Y_k|S_k$ is linked to the observation function g_k. For exemple, if this function is linear and if the noise is Gaussian, the likelihood is also Gaussian:

$$Y_k|S_k \sim \mathcal{N}\left(g_k(S_k), P_{n_k}\right) \tag{14}$$

If the mappings are linear and if the noises n_k and v_k are Gaussian, prior and posterior distributions are analytically computable and the optimal solution is given by the Kalman filter [15]: quantities of interest are random variables, and inference (that is prediction of these quantities and correction of them given a new observation) is done online by propagating sufficient statistics through linear transformations. If the

mappings are nonlinear (but the noises are still Gaussian), a first solution is to linearize them around the state: it is the principle of the Extended Kalman Filter (EKF), and sufficient statistics are still propagated through linear transformations. Another approach is the Sigma Point Kalman Filter (SPKF) framework [6]. The basic idea is that it is easier to approximate a probability distribution than an arbitrary nonlinear function. It is based on the unscented transform [16], which is now described (the sigma-point designation can be seen as a generalization of the unscented transform).

2) The Unscented Transform: The problem of approximating Bayesian filtering when evolution and observation equations are not linear can be expressed as follows: given first and second order moment of a random variable, compute first and second order moments of a nonlinear mapping of this random variable. The unscented transform addresses this issue by deterministically sampling the distribution using its mean and variance.

Let's abstract from previous sections and their notations. Let X be a random vector, and let Y be a mapping of X. The problem is to compute mean and covariance of Y knowing the mapping and first and second order moments of X. If the mapping is linear, the relation between X and Y can be written as $Y = AX$ where A is a matrix of *ad hoc* dimension. In this case, required mean and covariance can be analytically computed: $E[Y] = AE[X]$ and $E[YY^T] = AE[XX^T]A^T$.

If the mapping is nonlinear, the relation between X and Y can be generically written as:

$$Y = f(X) \tag{15}$$

A first solution would be to approximate the nonlinear mapping, that is to linearize it around the mean of the random vector X. This leads to the following approximations of the mean and covariance of Y:

$$E[Y] \approx f\left(E[X]\right) \tag{16}$$

$$E[YY^T] \approx \left(\nabla f\left(E[X]\right)\right) E[XX^T] \left(\nabla f\left(E[X]\right)\right)^T \tag{17}$$

This approach is the basis of Extended Kalman Filtering (EKF) [17], which has been extensively studied and used in past decades. However it has some limitations. First it cannot handle non-derivable nonlinearities. It requires to compute the gradient of the mapping f, which can be quite difficult even if possible. It also supposes that the nonlinear mapping is locally linearizable, which is unfortunately not always the case and can lead to quite bad approximations, as exemplified in [16].

The basic idea of the unscented transform is that it is easier to approximate an arbitrary random vector than an arbitrary nonlinear function. Its principle is to sample *deterministically* a set of so-called sigma-points from the expectation and the covariance of X. The images of these points through the nonlinear mapping f are then computed, and they are used to approximate statistics of interest. It shares similarities with Monte-Carlo methods, however here the sampling is deterministic and requires less samples to be drawn, nonetheless allowing a given accuracy [16].

The original unscented transform is now described more formally (some variants have been introduced since, but the basic principle is the same). Let n be the dimension of X. A set of $2n+1$ sigma-points is computed as follows:

$$x_0 = \bar{X} \qquad j = 0 \tag{18}$$

$$x_j = \bar{X} + \left(\sqrt{(n+\kappa)P_X}\right)_j \qquad 1 \leq j \leq n \tag{19}$$

$$x_j = \bar{X} - \left(\sqrt{(n+\kappa)P_X}\right)_{n-j} \qquad n+1 \leq j \leq 2n \tag{20}$$

as well as associated weights:

$$w_0 = \frac{\kappa}{n+\kappa} \text{ and } w_j = \frac{1}{2\left(n+\kappa\right)} \forall j > 0 \tag{21}$$

where $\bar{X}$ is the mean of X, P_X is its variance matrix, κ is a scaling factor which controls the accuracy of the unscented transform [16], and $(\sqrt{(n+\kappa)P_X})_j$ is the j^{th} column of the Cholesky decomposition of the matrix $(n+\kappa)P_X$. Then the image through the mapping f is computed for each of these sigma-points:

$$y_j = f(x_j), \quad 0 \leq j \leq 2n \tag{22}$$

The set of sigma-points and their images can finally be used to compute first and second order moments of Y, and even P_{XY}, the covariance matrix between X and Y:

$$\bar{Y} \approx \bar{y} = \sum_{j=0}^{2n} w_j y_j \tag{23}$$

$$P_Y \approx \sum_{j=0}^{2n} w_j \left(y_j - \bar{y}\right) \left(y_j - \bar{y}\right)^T \tag{24}$$

$$P_{XY} \approx \sum_{j=0}^{2n} w_j \left(x_j - \bar{x}\right) \left(y_j - \bar{y}\right)^T \tag{25}$$

where $\bar{x} = x_0 = \bar{X}$.

3) Sigma Point Kalman Filtering: The unscented transform having been presented, the Sigma-Point Kalman Filtering, which is an approximation of Bayesian filtering for nonlinear mapping based on unscented and similar transforms (*e.g.*, central differences transform, see [6] for details), is shortly described.

SPKF and classical Kalman equations are very similar. The major change is how to compute sufficient statistics (directly for Kalman, through sigma points for SPKF). Algorithm 2 sketches a SPKF update based on the state-space formulation (10), and using the standard Kalman notations: $s_{k|k-1}$ denotes a prediction, $s_{k|k}$ an estimate (or correction), $P_{s,y}$ a covariance matrix, $\bar{n}_k$ a mean and k is the discrete time index. The principle of each update is as follow. First, the prediction step consists in predicting the current mean and covariance for the hidden state given previous estimates and using the evolution equation. From this and using the observation equation, the current observation is predicted. This implies to use the unscented transform if the mappings are nonlinear. Then mean and covariance of the hidden state are corrected using the current observation and some system

statistics (the better the prediction, the lesser the correction). Update (or equivalently correction step) is made according to the so-called Kalman gain K_k which depends on statistics computed thanks to the unscented transform and using system dynamics. Notice also that being online this algorithm must be initialized with some priors $\bar{s}_{0|0}$ and $P_{0|0}$.

The reader can refer to [6] for details. More precise algorithms will be given later, when developing specific approaches. Generally speaking, the computational complexity of such an update is $O(|S|^3)$, where $|S|$ is the dimension of the state space. However, it will be shown that a square computational complexity is possible for the approach developed in next sections.

Algorithm 2: SPKF Update

inputs : $\bar{s}_{k-1|k-1}$, $P_{k-1|k-1}$
outputs: $\bar{s}_{k|k}$, $P_{k|k}$

Sigma-points computation;
Compute deterministically sigma-point set $S_{k-1|k-1}$ from $\bar{s}_{k-1|k-1}$ and $P_{k-1|k-1}$;

Prediction Step;
Compute sigma-point set $S_{k|k-1}$ from $f_k(S_{k-1|k-1}, \bar{n}_k)$ and process noise covariance;
Compute $\bar{s}_{k|k-1}$ and $P_{k|k-1}$ from $S_{k|k-1}$;

Correction Step;
Observe y_k;
Compute sigma-point set $Y_{k|k-1} = g_k(S_{k|k-1}, \bar{v}_k)$;
Compute $\bar{y}_{k|k-1}$, P_{y_k} and P_{sy_k} from $S_{k|k-1}$, $Y_{k|k-1}$ and observation noise covariance;
$K_k = P_{sy_k} P_{y_k}^{-1}$;
$\bar{s}_{k|k} = \bar{s}_{k|k-1} + K_k(y_k - \bar{y}_{k|k-1})$;
$P_{k|k} = P_{k|k-1} - K_k P_{y_k} K_k^T$;

III. Supervised Learning

The approach presented in this section addresses the problem of nonlinear function approximation. The aim here is to approximate a nonlinear function $f(x)$, $x \in \mathcal{X}$, where $\mathcal{X}$ is a compact set of $\mathbb{R}^n$, from noisy samples

$$(x_k, y_k = f(x_k) + n_k)_k \tag{26}$$

where k is the time index and n_k is the observation random noise, by a function $\hat{f}_\theta(x)$ parameterized by the vector θ. The rest of this paper is written for a scalar output, nevertheless Bayesian filtering paradigm allows an easy extension to the vectorial output case.

The hint is to cast this regression problem in a state-space formulation. The parameter vector is considered as the hidden state to be infered. It is thus modeled as a random variable. As a process model, a random walk is chosen, which is generally a good choice if no more information is available. First it allows handling non-stationarity, but it can also help to avoid local minima, as discussed before. The observation equation links the observations (noisy samples from the function of interest) to the parameterized function. Notice that even if the function of interest is not noisy, it does not necessarily exist in the function space spanned by the parameters, so the observation noise is also structural. This gives the following state-space formulation :

$$\begin{cases} \theta_{k+1} = \theta_k + v_k \\ y_k = \hat{f}_{\theta_k}(x_k) + n_k \end{cases} \tag{27}$$

As announced in Section I, the principle of the propose approach is to choose a (nonlinear) kernel representation of the value function, to use the dictionary method to automate the choice of the structure, and to use a SPKF to track the parameters.

A. Parameterization

The approximation is of the form

$$\hat{f}_\theta(x) = \sum_{i=1}^{p} \alpha_i K_{\sigma_i}(x, x_i) \tag{28}$$

$$\text{where } \theta = [(\alpha_i)_{i=1}^p,\ (x_i)_{i=1}^p,\ (\sigma_i)_{i=1}^p]^T$$

$$\text{and } K_{\sigma_i}(x, x_i) = \exp(-\frac{\|x - x_i\|^2}{2\sigma_i^2})$$

The problem is to find the optimal number of kernels and a good initialisation for the parameters (extension to other types of kernel is quite straightforward). As a preprocessing step, the dictionary method of Section II-B is used. First a prior σ_0 is put on the Gaussian width (or alternatively on *ad hoc* parameters if another kernel is considered, *e.g.*, degree and bias for a polynomial kernel). Then N random points are sampled uniformly from $\mathcal{X}$ and used to compute the dictionary. Thus a set of p points $\mathcal{D} = \{x_1, \ldots, x_p\}$ is obtained such that

$$\varphi_{\sigma_0}(\mathcal{X}) \simeq \text{Span}\,\{\varphi_{\sigma_0}(x_1), \ldots, \varphi_{\sigma_0}(x_p)\} \tag{29}$$

where φ_{σ_0} is the mapping corresponding to the kernel K_{σ_0}. Notice that even if the sparsification procedure is offline, the algorithm (the regression part) is online. Moreover, no training sample is needed for this preprocessing step, but only classical prior which is anyway needed for the Bayesian filter (σ_0), one sparsification parameter ν and bounds for $\mathcal{X}$. These requirements are not too restrictive.

Let q be the number of parameters. Given the chosen parameterization of equation (28), there is p parameters for the weights, p parameters for Gaussian standard deviations and np parameters for Gaussian centers: $q = (2 + n)p$.

B. Prior

As for any Bayesian approach (and more generally any online method) a prior (an initialization) has to be put on the (supposed Gaussian) parameter distribution:

$$\theta_0 \sim \mathcal{N}(\bar{\theta}_0, \Sigma_{\theta_0}) \tag{30}$$

where mean and covariance matrix are defined as

$$\begin{cases} \bar{\theta}_0 = [\alpha_0, \ldots, \alpha_0, \mathcal{D}, \sigma_0, \ldots, \sigma_0]^T \\ \Sigma_{\theta_0} = \text{diag}([\sigma_{\alpha_0}^2, \ldots, \sigma_{\alpha_0}^2, \sigma_{\mu_0}^2, \ldots, \sigma_{\mu_0}^2, \sigma_{\sigma_0}^2, \ldots, \sigma_{\sigma_0}^2]) \end{cases} \tag{31}$$

The operator diag applied to a column vector gives a diagonal matrix. In these expressions, α_0 is the prior mean on kernel weights, $\mathcal{D}$ is the dictionary computed in the preprocessing step, σ_0 is the prior mean on kernel deviation, and $\sigma^2_{\alpha_0}$, $\sigma^2_{\mu_0}$, $\sigma^2_{\sigma_0}$ are respectively the prior variance on kernel weights, centers and deviations. All these parameters (except the dictionary) have to be set up beforehand. Notice that $\bar{\theta}_0 \in \mathbb{R}^q$ and $\Sigma_{\theta_0} \in \mathbb{R}^{q\times q}$. A prior has also to be put on noises: $v_0 \sim \mathcal{N}(0, R_{v_0})$ where $R_{v_0} = \sigma^2_{v_0} I_q$, I_q being the identity matrix, and $n_0 \sim \mathcal{N}(0, R_{n_0})$, where $R_{n_0} = \sigma^2_{n_0}$ is a scalar.

C. Artificial Process Noise

Another issue is to choose the artificial process noise. Formally, since the target function is stationary, there is no process noise. However introducing an artificial process noise can strengthen convergence and robustness properties of the filter. Choosing this noise is still an open research problem. Following [6] two types of artificial process noise are considered (the observation noise is chosen to be constant).

The first one is a Robbins-Monro stochastic approximation scheme for estimating innovation. That is the process noise covariance is set to

$$R_{v_k} = (1-\alpha_{RM})R_{v_{k-1}} + \alpha_{RM} K_k (y_k - \hat{f}_{\bar{\theta}_{k|k}}(x_k))^2 K_k^T \quad (32)$$

Here α_{RM} is a forgetting factor set by the user, and K_k is the Kalman gain obtained during the Bayesian filter update.

The second type of noise provides for an approximate exponentially decaying weighting past data. Its covariance is set to a fraction of the parameters covariance, that is

$$R_{v_k} = (\lambda^{-1} - 1) P_{k|k} \quad (33)$$

where $\lambda \in]0, 1]$ $(1 - \lambda \ll 1)$ is a forgetting factor similar to the one from the recursive least-squares (RLS) algorithm.

D. Tracking Parameters

A Sigma Point Kalman filter update (which updates first and second order moments of the random parameter vector) is applied at each time step, as a new training sample (x_k, y_k) is available. The evolution equation being linear, the update algorithm does not involve the computation of a sigma-point set in the prediction step as in Algorithm 2. The proposed approach is fully described in Algorithm 3.

First, the dictionary has to be computed and priors have to be chosen. Then, as each input-output couple (x_k, y_k) is observed, the parameter vector mean and covariance are updated. First is the prediction phase. As the parameter vector evolution is modeled as a random walk, the prediction of the mean at time k is equal to the estimate of this mean at time $k-1$ (that is $\bar{\theta}_{k|k-1} = \bar{\theta}_{k-1|k-1}$), and the parameters covariance is updated thanks to the process noise covariance. Then, the set of $2q+1$ sigma-points $\Theta_{k|k-1}$ corresponding to the parameters distribution must be computed using $\bar{\theta}_{k|k-1}$ and $P_{k|k-1}$, as well as associated weights $\mathcal{W}$ (see Section II-C2):

$$\Theta_{k|k-1} = \left\{ \theta^{(j)}_{k|k-1}, \quad 0 \le j \le 2q \right\} \quad (34)$$

$$\mathcal{W} = \{ w_j, \quad 0 \le j \le 2q \quad \} \quad (35)$$

Notice that each of these sigma-points corresponds to a particular parameterized function. Each of these functions is evaluated in x_k, the current observed input, which forms the set of images of the sigma-points:

$$\mathcal{Y}_{k|k-1} = \left\{ y^{(j)}_{k|k-1} = \hat{f}_{\theta^{(j)}_{k|k-1}}(x_k),\ 0 \le j \le 2q \right\} \quad (36)$$

It can be seen as an approximated sampled prior distribution over observations. This sigma-point set and its image are then used to compute the prediction of the observation as well as some statistics necessary to the computation of the Kalman gain:

$$\bar{y}_{k|k-1} = \sum_{j=0}^{2q} w_j y^{(j)}_{k|k-1} \quad (37)$$

$$P_{\theta y_k} = \sum_{j=0}^{2q} w_j (\theta^{(j)}_{k|k-1} - \bar{\theta}_{k|k-1})(y^{(j)}_{k|k-1} - \bar{y}_{k|k-1}) \quad (38)$$

$$P_{y_i} = \sum_{j=0}^{2q} w_j \left(y^{(j)}_{k|k-1} - \bar{y}_{k|k-1} \right)^2 + P_{n_k} \quad (39)$$

Finally, the Kalman gain can be computed, and the estimated mean and covariance can be updated:

$$K_k = P_{\theta y_k} P^{-1}_{y_k} \quad (40)$$

$$\bar{\theta}_{k|k} = \bar{\theta}_{k|k-1} + K_k \left(y_k - \bar{y}_{k|k-1} \right) \quad (41)$$

$$P_{k|k} = P_{k|k-1} - K_k P_{y_k} K_k^T \quad (42)$$

If necessary, the artificial process noise is updated following one of the two proposed schemes. A constant or null process noise can also be considered.

Notice that practically a square-root implementation of this algorithm is used, which principally avoids a full Cholesky decomposition at each time-step. This approach is computationally cheaper: the complexity per iteration is $O(q^2)$, where it is recalled that $q = |\theta|$ is the size of the parameter vector ($O(q^3)$ in the general case). See [6] for details concerning this square-root implementation of SPKF.

E. Experiments

In this section, the aim is to experiment Algorithm 3 by regressing a cardinal sine ($\mathrm{sinc}(x) = \frac{\sin(x)}{x}$) on $\mathcal{X} = [-10, 10]$. It is an easy problem, but a common benchmark too which allows comparison to state-of-the-art algorithms. The uncertainty about function approximation which can be computed from this framework is also illustrated.

1) Problem Statement and Settings: At each time step k samples $x_k \sim \mathcal{U}_\mathcal{X}$ (x_k uniformly sampled from $\mathcal{X}$) and $y_k = \mathrm{sinc}(x_k) + w_k$ where $w_k \sim \mathcal{N}(0, \sigma^2_w)$ are observed. Notice that the true observation covariance noise σ^2_w and the prior σ^2_v are distinguished. For those experiments the first type of artificial process noise presented in Section III-C is used. The true noise is set to $\sigma_w = 0.1$. The algorithm parameters are set to $N = 100$, $\sigma_0 = 1.6$, $\nu = 0.1$, $\alpha_0 = 0$, $\sigma_v = 0.5$, $\alpha_{RM} = 0.7$, and all variances ($\sigma^2_{\alpha_0}$, $\sigma^2_{\mu_0}$, $\sigma^2_{\sigma_0}$, $\sigma^2_{n_0}$) to 0.1. Notice that this parameters were not finely tuned (only orders of magnitude seem to be important).

Algorithm 3: Direct regression algorithm

Compute dictionary;
$\forall i \in \{1 \dots N\},\ x_i \sim \mathcal{U}_{\mathcal{X}}$;
Set $X = \{x_1, \dots, x_N\}$;
$\mathcal{D}$ =`Compute-Dictionary` (X,ν,σ_0) ;

Initialisation;
Initialise $\bar{\theta}_0$, $P_{0|0}$, R_{n_0}, R_{v_0};

for $k = 1, 2, \dots$ **do**

Observe $(\mathbf{x_k}, \mathbf{y_k})$;

SPKF update;

Prediction Step;
$\bar{\theta}_{k|k-1} = \bar{\theta}_{k-1|k-1}$;
$P_{k|k-1} = P_{k-1|k-1} + P_{v_{k-1}}$;

Sigma-points computation ;
$\Theta_{k|k-1} = \left\{ \theta^{(j)}_{k|k-1}, \quad 0 \leq j \leq 2q \right\}$;
$\mathcal{W} = \{ w_j, \quad 0 \leq j \leq 2q \quad \}$;
$\mathcal{Y}_{\mathbf{k|k-1}} = \left\{ \mathbf{y^{(j)}_{k|k-1}} = \hat{\mathbf{f}}_{\theta^{(\mathbf{j})}_{\mathbf{k|k-1}}}(\mathbf{x_k}),\ \mathbf{0 \leq j \leq 2q} \right\}$;

Compute statistics of interest;
$\bar{y}_{k|k-1} = \sum_{j=0}^{2q} w_j y^{(j)}_{k|k-1}$;
$P_{\theta y_k} = \sum_{j=0}^{2q} w_j (\theta^{(j)}_{k|k-1} - \bar{\theta}_{k|k-1})(y^{(j)}_{k|k-1} - \bar{y}_{k|k-1})$;
$P_{y_i} = \sum_{j=0}^{2q} w_j \left(y^{(j)}_{k|k-1} - \bar{y}_{k|k-1} \right)^2 + P_{n_k}$;

Correction step;
$K_k = P_{\theta y_k} P^{-1}_{y_k}$;
$\bar{\theta}_{k|k} = \bar{\theta}_{k|k-1} + K_k \left(y_k - \bar{y}_{k|k-1} \right)$;
$P_{k|k} = P_{k|k-1} - K_k P_{y_k} K_k^T$;

Artificial process noise update;
Robbins-Monro covariance;
$R_{v_k} =$
$(1 - \alpha_{RM}) R_{v_{k-1}} + \alpha_{RM} K_k (y_k - \hat{f}_{\bar{\theta}_{k|k}}(x_k))^2 K_k^T$;
or;
Forgetting covariance;
$R_{v_k} = (\lambda^{-1} - 1) P_{k|k}$;

2) Quality of Regression: The quality of regression is measured with the RMSE (Root Mean Square Error) $\|f - \hat{f}_\theta\|$ (where $\|.\|$ is the usual L_2-norm), computed over 200 equally-spaced points. Averaged over 100 runs of Algorithm 3, a RMSE of 0.0676 ± 0.0176 is obtained for 50 samples, of 0.0452 ± 0.0092 for 100 samples, and of 0.0397 ± 0.0065 for 200 samples, for an average of 9.6 kernels. This is illustrated on Figure 1. A typical result is given in Figure 2 (50 observed samples). The dotted line, solid line and the crosses represent respectively the cardinale sine, the regression and the observations.

The proposed algorithm compares favorably with state-of-the-art batch and online algorithms. Table I shows the performance of the proposed algorithm and of other methods (some being online and other batch, some handling uncertainty information and other not), for which results are reproduced

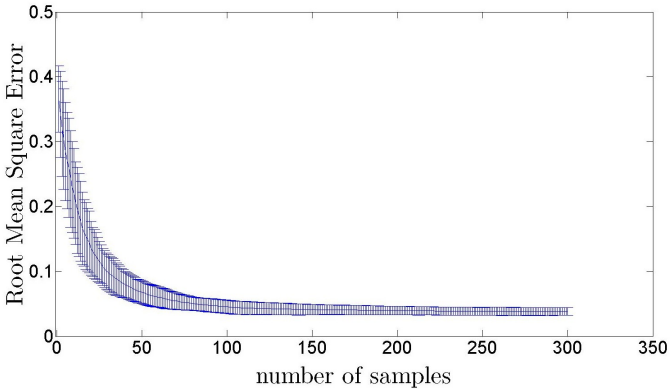


Fig. 1. RMSE (mean ± deviation).

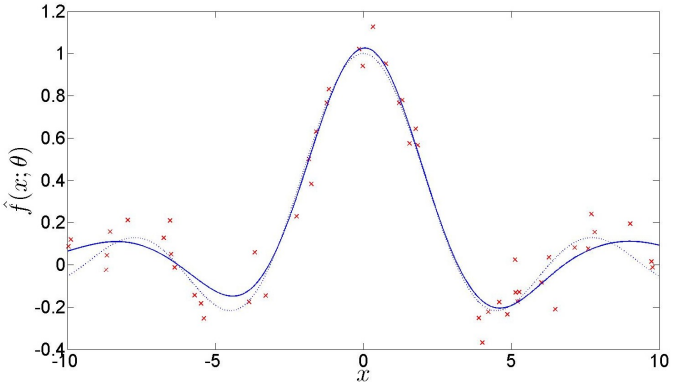


Fig. 2. Typical regression.

from [8] (see this paper and references therein for details about these algorithms). For each method is given the RMSE as well as the number of kernel functions, and the associated variations, the SVM (Support Vector Machine) [4] being the baseline. The proposed method achieves the best RMSE with slightly more kernels than other approaches. However the sparsification parameter ν of the dictionary can be tuned in order to address the trade-off between number of kernels and quality of approximation. Moreover, recall that the parameters were not finely tuned.

3) Uncertainty of Generalization: Through the sigma point approach, it is also possible to derive a confidence interval over $\mathcal{X}$. This allows to quantify the uncertainty of the regression at any point (and not a global upper bound as it is often computed

Method	test error	# kernels
Proposed algorithm	**0.0385 (-25.8%)**	**9.6 (-65.7%)**
Figueiredo	0.0455 (-12.3%)	7.0 (-75%)
SVM	**0.0519 (-0.0%)**	**28.0 (-0%)**
RVM	0.0494 (-4.8%)	6.9 (-75.3%)
VRVM	0.0494 (-4.8%)	7.4 (-73.5%)
MCMC	0.0468 (-9.83%)	6.5 (-76.8%)
Sequential RVM	0.0591 (+13.8%)	4.5 (-83.9%)

TABLE I
COMPARATIVE RESULTS.

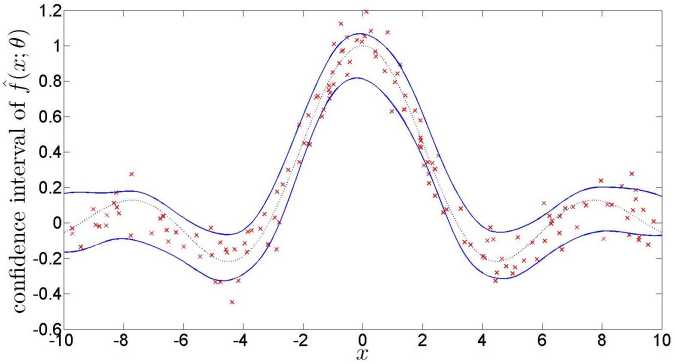


Fig. 3. Confidence (uniform distribution).

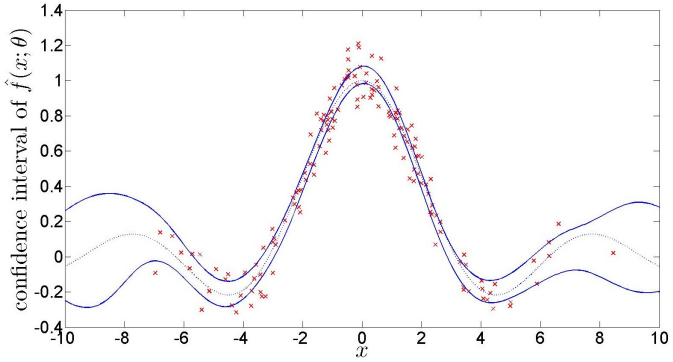


Fig. 4. Confidence (normal distribution).

in kernel-based regression methods). A typical confidence interval is illustrated on Figure 3. The dotted line, solid line and the crosses represent respectively the cardinale sine, the confidence interval (indeed the standard deviation, which corresponds to a confidence interval if Gaussian noise is assumed) and the observations. It can be particularly useful when the regression is used in a control framework, where this confidence approach can be used to take more cautious decisions (see [18] for example). It can be also useful for active learning, which aims at selecting costly samples such as gaining the more possible new information.

In Figure 3, the samples used to feed the regressor are sampled uniformly from $\mathcal{X}$, and thus the associated confidence interval has an approximately constant width. However, a regressor which handles uncertainty should do it locally. This is indeed the case for the proposed algorithm. In Figure 4, the distribution of samples is Gaussian zero-centered. It can be seen that the confidence interval is much larger where samples are less frequent (close to the bounds). So the computed confidence intervals make sens.

IV. Extension to the Case of Nonlinearly Mapped Observations

The problem addressed here is slightly different from the one in Section III. The function of interest is not directly observed, but only a nonlinear (and possibly non-derivable) mapping of it is available, the mapping being known analytically. Moreover, some of the involved nonlinearities can be just observed. This is the case when a special sensor has to be used (*e.g.*, measuring a temperature using a spectrometer or a thermocouple). This is also the case when solving the Bellman equation in a Markovian decision process with unknown deterministic transitions, which will be developped in Section V.

More formally, let $x = (x^1, x^2) \in \mathcal{X} = \mathcal{X}^1 \times \mathcal{X}^2$ where $\mathcal{X}^1$ (resp. $\mathcal{X}^2$) is a compact set of $\mathbb{R}^n$ (resp. $\mathbb{R}^m$). Let $t : \mathcal{X}^1 \times \mathcal{X}^2 \to \mathcal{X}^1$ be a nonlinear transformation (transitions in case of dynamic systems) which will be observed. Let g be a nonlinear mapping such that $g : f \in \mathbb{R}^{\mathcal{X}} \to g_f \in \mathbb{R}^{\mathcal{X}\times\mathcal{X}^1}$. The aim here is to approximate sequentially the nonlinear function $f : x \in \mathcal{X} \to f(x) = f(x^1, x^2) \in \mathbb{R}$ from samples

$$\left(x_k, t_k = t(x_k^1, x_k^2), y_k = g_f(x_k^1, x_k^2, t_k)\right)_k \tag{43}$$

by a function $\hat{f}_\theta(x) = \hat{f}_\theta(x^1, x^2)$ parameterized by the vector θ. Here the output is scalar, however the proposed method can be straightforwardly extended to vectorial outputs. Notice that as

$$\left(\mathbb{R}^{\mathcal{X}}\right)^{\mathbb{R}^{\mathcal{X}}} \subset \left\{ g \in \left(\mathbb{R}^{\mathcal{X}\times\mathcal{X}^1}\right)^{\mathbb{R}^{\mathcal{X}}} \; | g_f : x \in \mathcal{X} \to g_f(x, t(x)) \right\} \tag{44}$$

this problem statement is quite general. Thus the work presented in this section can be considered with $g : f \in \mathbb{R}^{\mathcal{X}} \to g_f \in \mathbb{R}^{\mathcal{X}}$ (g being known analytically), this case being more specific. The interest of this particular formulation is that a part of the nonlinearities has to be known analytically (the mapping g) whereas the other part can be just observed (the t function).

As before, this regression problem is stated in the corresponding state-space formulation:

$$\begin{cases} \theta_{k+1} = \theta_k + v_k \\ y_k = g_{\hat{f}_{\theta_k}}(x_k^1, x_k^2, t_k) + n_k \end{cases} \tag{45}$$

Recall that the algorithm proposed in Section III is derivative free and handle nonlinearities. Thus it is quite simple to adapt it to this new state-space formulation. A kernel-based representation is chosen as before, and its structure is automatized thanks to the dictionary method. A prior is put, and then a SPKF is used to learn the parameters.

A. Parameterization

Recall that the objective here is to sequentially approximate a nonlinear function, as samples $(x_k, t_k = t(x_k^1, x_k^2), y_k)$ are available, with a weighted sum of kernel functions. This parameter estimation problem can be expressed as the state-space problem (45). Notice that here f does not depend on time, however this approach can be easily extended to nonstationary function approximation. In this section, the approximation is

of the form

$$\hat{f}_\theta(x) = \sum_{i=1}^{p} \alpha_i K_{\sigma_i^1}(x^1, x_i^1) K_{\sigma_i^2}(x^2, x_i^2), \text{ with} \quad (46)$$

$$\theta = [(\alpha_i)_{i=1}^p, \left((x_i^1)^T\right)_{i=1}^p, (\sigma_i^1)_{i=1}^p, \left((x_i^2)^T\right)_{i=1}^p, (\sigma_i^2)_{i=1}^p]^T$$

$$\text{and } K_{\sigma_i^j}(x^j, x_i^j) = \exp(-\frac{\|x^j - x_i^j\|^2}{2(\sigma_i^j)^2}),\ j = 1, 2$$

Notice that $K_{\sigma_i}(x, x_i) = K_{(\sigma_i^1, \sigma_i^2)}\left((x^1, x^2), (x_i^1, x_i^2)\right) = K_{\sigma_i^1}(x^1, x_i^1) K_{\sigma_i^2}(x^2, x_i^2)$ is a product of kernels, thus it is a kernel. Again, extension to other kernels is straightforward. The optimal number of kernels and a good initialisation for the parameters have to be determined first.

To tackle this initial difficulty, the dictionary method is used in a preprocessing step. A prior $\sigma_0 = (\sigma_0^1, \sigma_0^2)$ on the Gaussian width is first put (the same for each kernel). Then a set of N random points is sampled uniformly from $\mathcal{X}$ and used to compute the dictionary. A set of p points $\mathcal{D} = \{x_1, \ldots, x_p\}$ is thus obtained such that $\varphi(\mathcal{X}) \simeq \text{Span}\{\varphi_{\sigma_0}(x_1), \ldots, \varphi_{\sigma_0}(x_p)\}$ where φ_{σ_0} is the mapping corresponding to the kernel K_{σ_0}. As before, even though this sparsification procedure is offline, the proposed algorithm (the regression part) is online.

B. Prior, Learning and Artificial Noises

A SPKF is then used to estimate the parameters. As for any Bayesian approach an initial prior on the parameter distribution has to be put. The initial parameter vector follows a Gaussian law, that is $\theta_0 \sim \mathcal{N}(\bar{\theta}_0, \Sigma_{\theta_0})$, where

$$\begin{cases} \bar{\theta}_0 = [\alpha_0, \ldots, \alpha_0, \mathcal{D}^1, \sigma_0^1, \ldots, \sigma_0^1, \mathcal{D}^2, \sigma_0^2, \ldots, \sigma_0^2]^T \\ \Sigma_{\theta_0} = \text{diag}(\sigma_{\alpha_0}^2, \ldots, \sigma_{\mu_0^1}^2, \ldots, \sigma_{\sigma_0^1}^2, \ldots, \sigma_{\mu_0^2}^2, \ldots, \sigma_{\sigma_0^2}^2, \ldots) \end{cases} \quad (47)$$

In these equations, α_0 is the prior mean on kernel weights, $\mathcal{D}^j = [(x_1^j)^T, \ldots, (x_p^j)^T]$ is derived from the dictionary computed in the preprocessing step, σ_0^j is the prior mean on kernel deviation, and $\sigma_{\alpha_0}^2$, $\sigma_{\mu_0^j}^2$ (which is a row vector with the variance for each component of x_i^j, $\forall i$) and $\sigma_{\sigma_0}^2$ are respectively the prior variances on kernel weights, centers and deviations. All these parameters (except the dictionary) have to be set up beforehand. Let $|\theta| = q = (n + m + 3)p$. Notice that $\bar{\theta}_0 \in \mathbb{R}^q$ and $\Sigma_{\theta_0} \in \mathbb{R}^{q \times q}$. A prior on noises has to be put too: more precisely, $v_0 \sim \mathcal{N}(0, R_{v_0})$ where $R_{v_0} = \sigma_{v_0}^2 I_q$ and $n \sim \mathcal{N}(0, R_n)$ where $R_n = \sigma_n^2$. Notice that here the observation noise is also structural. In other words, it models the ability of the parameterization $\hat{f}_\theta$ to approximate the true function of interest f. Then, a SPKF update is simply applied at each time step, as a new training sample (x_k, t_k, y_k) is available, as summarized in Algorithm 4. The only difference between this algorithm and Algorithm 3 is what is observed, as well as how to compute the set of images of sigma-points, directly using the approximated function $\hat{f}_\theta$ for the first one, and using its nonlinear mapping $g_{\hat{f}_\theta}$ for the second one.

A last issue is to choose the artificial process noise. The same noises as in Section III-C are considered, namely the Robbins-Monro stochastic approximation scheme for estimating innovation and the approximate exponentially decaying weighting past data. Alternatively a constant or null process noise can be considered. The observation noise is chosen constant.

Algorithm 4: Indirect regression algorithm

Compute dictionary;
$\forall i \in \{1 \ldots N\},\ x_i \sim \mathcal{U}_\mathcal{X}$;
Set $X = \{x_1, \ldots, x_N\}$;
$\mathcal{D}$ =`Compute-Dictionary` (X, ν, σ_0) ;

Initialisation;
Initialise $\bar{\theta}_0$, $P_{0|0}$, R_{n_0}, R_{v_0};

for $k = 1, 2, \ldots$ **do**

Observe $(\mathbf{x_k}, \mathbf{t_k}, \mathbf{y_k})$;

SPKF update;

Prediction Step;
$\bar{\theta}_{k|k-1} = \bar{\theta}_{k-1|k-1}$;
$P_{k|k-1} = P_{k-1|k-1} + P_{v_{k-1}}$;

Sigma-points computation ;
$\Theta_{k|k-1} = \left\{\theta_{k|k-1}^{(j)}, \quad 0 \le j \le 2q\right\}$;
$\mathcal{W} = \{w_j, \quad 0 \le j \le 2q \quad\}$;
$\mathcal{Y}_{\mathbf{k|k-1}} = \left\{\mathbf{y}_{\mathbf{k|k-1}}^{(\mathbf{j})} = \mathbf{g}_{\hat{\mathbf{f}}_{\boldsymbol{\theta}_{\mathbf{k|k-1}}^{(\mathbf{j})}}}(\mathbf{x_k}, \mathbf{t_k}),\ \mathbf{0 \le j \le 2q}\right\}$;

Compute statistics of interest;
$\bar{y}_{k|k-1} = \sum_{j=0}^{2q} w_j y_{k|k-1}^{(j)}$;
$P_{\theta y_k} = \sum_{j=0}^{2q} w_j (\theta_{k|k-1}^{(j)} - \bar{\theta}_{k|k-1})(y_{k|k-1}^{(j)} - \bar{y}_{k|k-1})$;
$P_{y_i} = \sum_{j=0}^{2q} w_j \left(y_{k|k-1}^{(j)} - \bar{y}_{k|k-1}\right)^2 + P_{n_k}$;

Correction step;
$K_k = P_{\theta y_k} P_{y_k}^{-1}$;
$\bar{\theta}_{k|k} = \bar{\theta}_{k|k-1} + K_k \left(y_k - \bar{y}_{k|k-1}\right)$;
$P_{k|k} = P_{k|k-1} - K_k P_{y_k} K_k^T$;

Artificial process noise update;
Robbins-Monro covariance;
$R_{v_k} =$
$(1 - \alpha_{RM}) R_{v_{k-1}} + \alpha_{RM} K_k (y_k - g_{\hat{f}_{\bar{\theta}_{k|k}}}(x_k, t_k))^2 K_k^T$;
or;
Forgetting covariance;
$R_{v_k} = (\lambda^{-1} - 1) P_{k|k}$;

C. Experiments

This section aims at illustrating Algorithm 4. The proposed artificial problem is quite arbitrary, it has been chosen for its interesting nonlinearities so as to emphasize on the potential benefits of the proposed approach. Let $\mathcal{X} = [-10, 10]^2$ and f be a 2d nonlinear cardinal sine:

$$f(x^1, x^2) = \frac{\sin(x^1)}{x^1} + \frac{x^1 x^2}{100} \quad (48)$$

Let the observed nonlinear transformation be

$$t(x^1, x^2) = 10 \tanh(\frac{x^1}{7}) + \sin(x^2) \quad (49)$$

Recall that this analytical expression is only used to generate observations in this experiment but it is not used in the regression algorithm. Let the nonlinear mapping g be

$$g_f\left(x^1, x^2, t(x^1, x^2)\right) = f(x^1, x^2) - \gamma \max_{y \in \mathcal{X}^2} f\left(t(x^1, x^2), y\right) \tag{50}$$

with $\gamma \in [0, 1[$ a predefined constant. This is a specific form of the Bellman equation [10] with the transformation t being a deterministic transition function. Recall that this equation is of special interest in optimal control theory. The associated state-space formulation is thus

$$\begin{cases} \theta_{k+1} = \theta_k + v_k \\ y_k = \hat{f}_{\theta_k}(x_k^1, x_k^2) - \gamma \max_{x^2 \in \mathcal{X}^2} \hat{f}_{\theta_k}\left(t(x_k^1, x_k^2), x^2\right) + n_k \end{cases} \tag{51}$$

1) Problem Statement and Settings: At each time step k the regressor observes $x_k \in \mathcal{U}_{\mathcal{X}}$ (x_k uniformly sampled from $\mathcal{X}$), the transformation $t_k = t(x_k^1, x_k^2)$ and $y_k = f(x_k) - \gamma \max_{x^2 \in \mathcal{X}^2} f(t_k, x^2)$. Once again the regressor does not have to know the function t analytically, it just has to observe it. The function f is shown on Figure 5(a) and its nonlinear mapping on Figure 5(b). For these experiments the first type of artificial process noise presented in Section III-C is used. The algorithm parameters were set to $N = 1000$, $\sigma_0^j = 4.4$, $\nu = 0.1$, $\alpha_0 = 0$, $\sigma_n = 0.05$, $\alpha_{RM} = 0.7$, and all variances ($\sigma^2_{\alpha_0}$, $\sigma^2_{\mu_0^j}$, $\sigma^2_{\sigma_0^j}$, $\sigma^2_{n_0}$) to 0.1, for $j = 1, 2$. The factor γ was set to 0.9. Notice that this empirical parameters were not finely tuned.

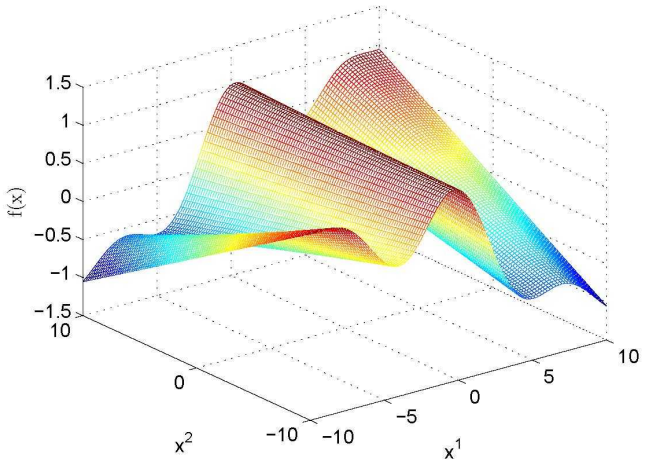
2) Quality of Regression: Because of the special form of g_f, any constant bias added to f still gives the same observations and it may exist other invariances: the root mean square error (RMSE) between f and $\hat{f}$ should not be used to measure the quality of regression. Instead the nonlinear mapping of f is computed from its estimate $\hat{f}$ and is then used to calculate the RMSE. The quality of regression is thus measured with the following RMSE:

$$\sqrt{\int_{\mathcal{X}} \left(g_f(x, t(x)) - g_{\hat{f}_\theta}(x, t(x))\right)^2 dx} \tag{52}$$

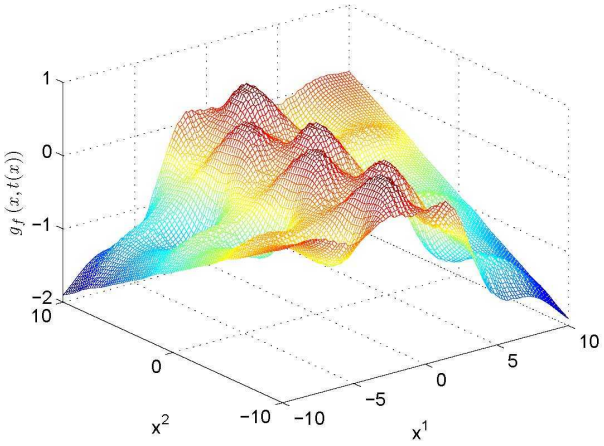
As it is computed from $\hat{f}$, it really measures the quality of regression, and as it uses the associated nonlinear mapping, it will not take into account the bias. Practically it is computed on 10^4 equally spaced points. The function g_f is what is observed (Figure 5(b)) and it is used by the SPKF to approximate f (Figure 5(a)) by $\hat{f}_\theta$ (Figure 6(a)). The nonlinear mapping of the approximated function $g_{\hat{f}_\theta}$ is computed from $\hat{f}_\theta$ (Figure 6(b)) and it is used to compute the RMSE.

As the proposed algorithm is stochastic, the results have been averaged over 100 runs. Figure 7 shows an errorbar plot, that is mean $\pm$ standard deviation of RMSE. The average number of kernels was 26.55 ± 1.17. This results can be compared with the RMSE directly computed from $\hat{f}$, that is

$$\sqrt{\int_{x \in \mathcal{X}} (f(x) - \hat{f}_\theta(x))^2 dx} \tag{53}$$



(a) 2-dimensional nonlinear cardinal sine.



(b) Nonlinear mapping observed by the regressor.

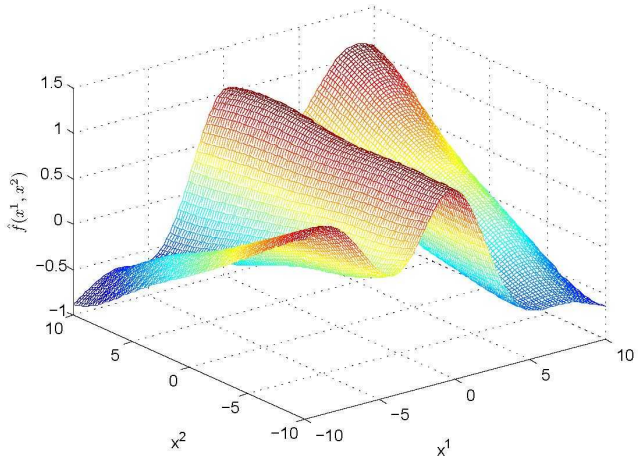
Fig. 5. Original functions.

	500	1000	1500
$g_{\hat{f}_\theta}$	0.072 ± 0.022	0.031 ± 0.005	0.024 ± 0.003
$\hat{f}_\theta$	0.296 ± 0.149	0.108 ± 0.059	0.066 ± 0.035

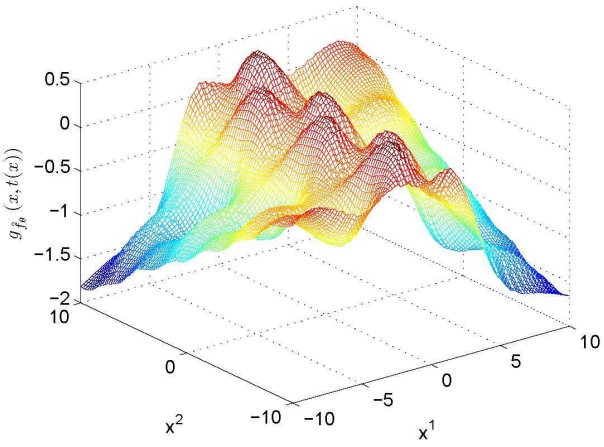
TABLE II
RMSE (MEAN $\pm$ DEVIATION).

which is illustrated on Table II (RMSE as a function of number of samples). There is more variance and bias in these results because of the possible invariances of the considered nonlinear mapping. However one can observe on Figure 6(a) that a good approximation is obtained.

Thus the RMSE computed from g_f is a better quality measure. As far as we know, no other method to handle such a problem has been published so far, thus comparisons with other approaches is made difficult. However the order of magnitude for the RMSE obtained from g_f is comparable with the state-of-the-art regression approaches when the function of interest is directly observed. This is demonstrated on Table III. Here the problem is to regress a linear 2d cardinal sine $f(x^1, x^2) = \frac{\sin(x^1)}{x^1} + \frac{x^2}{10}$. For the proposed algorithm, the



(a) Approximation of $f(x)$.



(b) Nonlinear mapping calculated from $\hat{f}_\theta(x)$.

Fig. 6. Approximated functions.

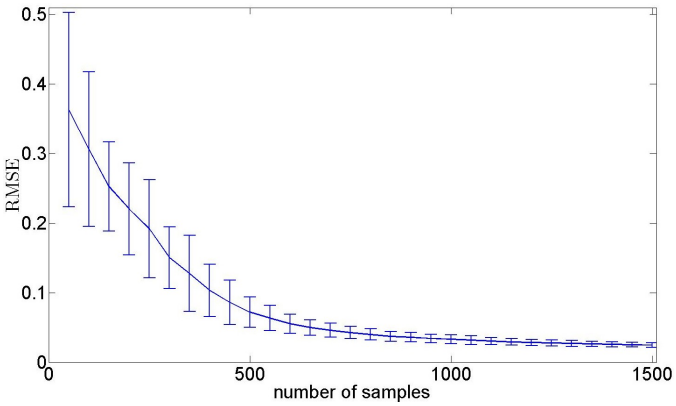


Fig. 7. RMSE (nonlinear mapping).

Method	test error $\hat{f}$	test error $g_{\hat{f}}$	%DV/SVs
Proposed Alg.	2.45×10^{-1}	4.5×10^{-2}	2.6%
KRLS	1.5×10^{-2}		7%
SVR	5.5×10^{-2}		60%

TABLE III
COMPARATIVE RESULTS.

nonlinear mapping is the same as considered before. RMSE results of Algorithm 4 are compared to Kernel Recursive Least Squares (KRLS) and Support Vector Regression (SVR). For the proposed approach, the approximation is computed from nonlinear mapping of observations. For KLRS and SVR, it is computed from direct observations. Notice that SVR is a batch method, that is it needs the whole set of samples in advance. The target of the regressor is indeed g_f, and the same order of magnitude is obtained (however much nonlinearities are introduced for this method and the representation is more sparse). The RMSE on $\hat{f}$ is slightly higher for this contribution, but this can be mostly explained by the invariances (as bias invariance) induced by the nonlinear mapping.

V. Application to Reinforcement Learning

The work presented in this section is a rather direct application of the general algorithm proposed in Section IV. However, before presenting the so-called *Bayesian Reward Filter*, the reinforcement learning paradigm is briefly introduced. Notice that the notion of state used in this section is different from the one used in Section II-C, despite same name and notation. Here the state is the state of a dynamic system to be controlled and not a parameter vector. Moreover it is directly observable and there is no need to infer it.

A. Reinforcement Learning Paradigm

Reinforcement learning (RL) [11] is a general paradigm in which an agent learns to control a dynamic system only through interactions. A feedback signal is provided to this agent as a reward information, which is a local hint about the quality of the control. Markov Decision Processes (MDP) are a common framework to solve this problem. A MDP is fully described by the state space that can be explored, the action set that can be chosen by the agent, a family of transition probabilities between states conditioned by the actions and a set of expected rewards associated to transitions. This is further explained in Section V-B. In this framework, at each time step k, the system adopts a state s_k. According to this, the agent can chose an action a_k, which leads to a transition to state s_{k+1} and to the obtention of a reward r_k, the agent objective being to maximize the future expected cumulative rewards. Here the knowledge of the environment is modelled as a Q-function which maps state action pairs to the expected cumulative rewards when following a given associated policy after the first transition. The proposed approach is model-free, no model of transitions and reward distributions is (directly or explicitly) learnt or known.

Solutions exist for the RL problem with discrete state and action spaces. However they generally do not scale up very well and cannot handle continuous state and/or action spaces. A wide variety of function approximation schemes have thus been applied to reinforcement learning (see [11] as a starting point). This is known as the generalization problem, and it is proposed here to handle it with a Bayesian filtering approach. The idea to use Bayesian filtering for reinforcement learning is not novel, but it has been surprisingly little studied. In [19] a modification of the linear quadratic Gaussian Kalman filter model is proposed, which allows the on-line estimation of optimal control (which is off-line for the classical one). In [20] Gaussian processes are used for reinforcement learning. This method can be understood as an extension of the Kalman filter to an infinite dimensional hidden state (the Gaussian process), but it can only handle (optimistic) policy-iteration-like update rules (because of the necessary linearity of the observation equation), contrarily to the proposed contribution, which can be seen as a nonlinear extension of the parametric case developed in [20] to a nonlinear and value-iteration-like scheme. In [21] a Kalman filter bank is used to find the parameters of a piecewise linear approximation of the value function.

B. Problem Statement

A Markov Decision Process (MDP) consists of a state space S, an action space A, a Markovian transition probability $p : S \times A \to \mathcal{P}(S)$ and a bounded reward function $r : S \times A \times S \to \mathbb{R}$. A policy is a mapping from state to action space: $\pi : S \to A$. At each time step k, the system is in a state s_k, the agent chooses an action $a_k = \pi(s_k)$, and the system is then driven in a state s_{k+1} following the conditional probability distribution $p(.|s_k, a_k)$. The agent receives the associated reward $r_k = r(s_k, a_k, s_{k+1})$. Its goal is to find the policy which maximizes the expected cumulative rewards, that is the quantity $E_\pi[\sum_{k\in\mathbb{N}} \gamma^k r(S_k, A_k, S_{k+1})|S_0 = s_0]$ for every possible starting state s_0, the expectation being over the state transitions taken upon executing π, where $\gamma \in [0, 1[$ is a discount factor.

A classical approach to solve this optimization problem is to introduce the Q-function defined as:

$$Q^\pi(s,a) = \int_S p(z|s,a)\Big(r(s,a,z) + \gamma Q^\pi(z,\pi(z))\Big)dz \quad (54)$$

It is the expected cumulative rewards by taking action a in state s and then following the policy π. The optimality criterion is to find the policy π^* (and associated Q^*) such that for every state s and for every policy π, $\max_{a\in A} Q^*(s,a) \geq \max_{a\in A} Q^\pi(s,a)$. The optimal Q-function Q^* satisfies the Bellman's equation:

$$Q^*(s,a) = \int_S p(z|s,a)\left(r(s,a,z) + \gamma \max_{b\in A} Q^*(z,b)\right)dz \quad (55)$$

In the case of discrete and finite action and state spaces, the Q-learning algorithm provides a solution to this problem. Its principle is to update a tabular approximation of the optimal Q-function after each transition (s, a, r, s'):

$$\hat{Q}(s,a) \leftarrow \hat{Q}(s,a) + \alpha\left(r + \gamma \max_{b\in A} \hat{Q}(s',b) - \hat{Q}(s,a)\right) \quad (56)$$

where α is a learning rate. An interesting fact is that the Q-learning is an off-policy algorithm, that is it allows to learn the optimal policy (from the learned optimal Q-function) while following a suboptimal one, given that it is sufficiently explorative. The proposed contribution can be seen as an extension of this algorithm to a Bayesian filtering framework (however with other advantages). See [11] for a comprehensive introduction to reinforcement learning, or [22] for a more formal treatment.

The reward is what is observed, the Q function is what is searched, and both are linked by the Bellman equation. Suppose that the Q-function is parameterized (either linearly or nonlinearly) by a vector θ. The aim is to find a good approximation $\hat{Q}_\theta$ of the optimal Q-function Q^* by observing transitions (s, a, r, s'). This reward regression problem is cast into a state-space representation. For an observed transition (s_k, a_k, r_k, s'_k), it is written as:

$$\begin{cases} \theta_{k+1} = \theta_k + v_k \\ r_k = \hat{Q}_{\theta_k}(s_k, a_k) - \gamma \max_{a\in A} \hat{Q}_{\theta_k}(s'_k, a) + n_k. \end{cases} \quad (57)$$

Here v_k is the artificial process noise and n_k the centered observation noise including all the stochasticity of the MDP. The framework is thus posed, but is far from being solved. The observation equation is nonlinear and even non-derivable (because of the $\max$ operator), that is why classical methods such as the standard Kalman filter cannot be used. Formally, the process noise is null, nevertheless introducing an artificial noise can improve the stability and convergence performances of the filter, as discussed before. This can be seen as a special case of the algorithm of the previous section, which is developed next.

C. Algorithm

As before Gaussian kernels are chosen, and their mean and deviation are considered as parameters:

$$\hat{Q}_\theta(s,a) = \sum_{i=1}^{p} \alpha_i K_{\sigma_i^s}(s, s_i) K_{\sigma_i^a}(a, a_i) \quad (58)$$

$$\text{with } K_{\sigma_i^x}(x, x_i) = \exp\left(-(x - x_i)^T (\Sigma_i^x)^{-1}(x - x_i)\right),$$

$$\text{where } x = s, a,\ \Sigma_i^x = \text{diag}(\sigma_i^x)^2,$$

$$\text{and } \theta = [(\alpha_i)_{i=1}^p, (s_i^T)_{i=1}^p, (a_i^T)_{i=1}^p, ((\sigma_i^s)^T)_{i=1}^p, ((\sigma_i^a)^T)_{i=1}^p]^T$$

Once again, the dictionary method is used to automatize the choice of the structure for this kernel parameterization and the *ad hoc* prior is chosen. Once the parameters are initialized, the parameter vector has still to be updated as new observations (s_k, a_k, r_k, s'_k) are available. A SPKF is used, and this is a special case of the theory developed in Section IV, the

functions t and g being given by:

$$t : S \times A \to S \quad (59)$$
$$s, a \mapsto s'$$

and

$$g : \mathbb{R}^{S\times A} \to \mathbb{R}^{S\times A\times S} \quad (60)$$
$$Q \mapsto \left(s, a, s' \mapsto Q(s,a) - \gamma \max_{b\in A} Q(s', b) \right)$$

Thus Algorithm 4 can be specialized for reinforcement learning, which gives the Bayesian Reward Filter summarized in Algorithm 5.

Algorithm 5: Bayesian Reward Filter

Compute dictionary;
$\forall i \in \{1 \dots N\},\ [s_i, a_i]^T \sim \mathcal{U}_{S\times A}$;
Set $X = \{[s_1, a_1]^T, \dots, [s_N, a_N]^T\}$;
$\mathcal{D}$ =`Compute-Dictionary` ($S \times A$,ν,σ_0) ;

Initialisation;
Initialise $\bar{\theta}_0$, $P_{0|0}$, R_{n_0}, R_{v_0};

for $k = 1, 2, \dots$ **do**
Observe (s_k, a_k, s'_k, r_k);

SPKF update;

Prediction Step;
$\bar{\theta}_{k|k-1} = \bar{\theta}_{k-1|k-1}$;
$P_{k|k-1} = P_{k-1|k-1} + P_{v_{k-1}}$;

Sigma-points computation ;
$\Theta_{k|k-1} = \left\{ \theta^{(j)}_{k|k-1}, \quad 0 \le j \le 2q \right\}$;
$\mathcal{W} = \{ w_j, \quad 0 \le j \le 2q \quad \}$;
$\mathcal{R}_{k|k-1} = \Big\{ r^{(j)}_{k|k-1} = \hat{Q}_{\theta^{(j)}_{k|k-1}}(s_k, a_k) -$
$\gamma \max_{b\in A} \hat{Q}_{\theta^{(j)}_{k|k-1}}(s'_k, b),\ 0 \le j \le 2q \Big\}$;

Compute statistics of interest;
$\bar{r}_{k|k-1} = \sum_{j=0}^{2q} w_j r^{(j)}_{k|k-1}$;
$P_{\theta r_k} = \sum_{j=0}^{2q} w_j (\theta^{(j)}_{k|k-1} - \bar{\theta}_{k|k-1})(r^{(j)}_{k|k-1} - \bar{r}_{k|k-1})$;
$P_{r_i} = \sum_{j=0}^{2q} w_j \left(r^{(j)}_{k|k-1} - \bar{r}_{k|k-1} \right)^2 + P_{n_k}$;

Correction step;
$K_k = P_{\theta y_k} P_{y_k}^{-1}$;
$\bar{\theta}_{k|k} = \bar{\theta}_{k|k-1} + K_k \left(r_k - \bar{r}_{k|k-1} \right)$;
$P_{k|k} = P_{k|k-1} - K_k P_{y_k} K_k^T$;

Artificial process noise update;
Robbins-Monro covariance;
$R_{v_k} = (1 - \alpha_{RM}) R_{v_{k-1}} + \alpha_{RM} K_k (r_k +$
$\gamma \max_b \hat{Q}_{\bar{\theta}_{k|k}}(s'_k, b) - \hat{Q}_{\bar{\theta}_{k|k}}(s_k, a_k))^2 K_k^T$;
or;
Forgetting covariance;
$R_{v_k} = (\lambda^{-1} - 1) P_{k|k}$;

D. Maximum over action space

Notice that a technical difficulty can be to compute the maximum over the actions for the parameterized Q-function. This computation is necessary for the filter update. If the action space is discrete and finite, computation of the $\max$ is easy. Otherwise it is an optimization problem. A first solution could be to sample the action space and to compute the maximum over the obtained samples. However this is especially computationally inefficient. The used method is close to one proposed in [23].

The maximum over action kernel centers is computed: $\mu^a = \operatorname{argmax}_{a_i} \hat{Q}_\theta(s, a_i)$. It serves then as the initialization for the Newton-Raphson method used to find the maximum :

$$a_m \leftarrow a_m - \left((\nabla_a \nabla_a^T) \hat{Q}_\theta(s,a) \right)^{-1}_{a=a_m} \nabla_a \hat{Q}_\theta(s,a) \Big|_{a=a_m}$$

If the Hessian matrix is singular, a gradient ascent/fixed point scheme is used:

$$a_m \leftarrow a_m + \nabla_a \hat{Q}_\theta(s,a) \Big|_{a=a_m}$$

The obtained action a_m is considered as the action which maximizes the parameterized Q-function.

E. Experiments

The proposed Bayesian Reward Filter is demonstrated on two problems. First, the "wet-chicken" task is a continuous state and action space and stochastic problem. Second, the "mountain car" problem is a continuous state, discrete action and deterministic problem. The latter one requires an hybrid parametrization. Let's first discuss the choice of parameters. They are given for reproducibility, nevertheless the reader can skip this without hurting understanding.

1) Choice of Parameters: For both tasks the reinforcement learning discount factor is set to $\gamma = 0.9$. The dictionary sparsity factor is set to $\nu = 0.9$. The second type of artificial process noise presented in Section III-C is used, and similarly to the recursive least-squares algorithm, the adaptive process noise covariance is set to a high value, such that $\lambda^{-1} - 1 \simeq 10^{-6}$.

The initial choice of kernel deviations is problem dependant. However a practical good choice seems to take a fraction of the quantity $x(j)_{\max} - x(j)_{\min}$ for the kernel deviation associated to $x(j)$, the j^{th} component of the column vector x, $x(j)_{\max}$ and $x(j)_{\min}$ being the bounds on the values taken by the variable $x(j)$. The prior kernel weights are supposed to be centered, and the associated standard deviations are set to a little fraction of the theoretical bound on Q-function, that is $\frac{r_{\max}}{1-\gamma}$. Because of geometry of Gaussian distributions, centers provided by the dictionary are supposed to be approximately uniformly distributed, and the prior deviation of the j^{th} component of the vector x is set to a little fraction of $(x(j)_{\max} - x(j)_{\min}) p^{-\frac{1}{ns+na}}$, with the convention that for discrete spaces $n = 0$. Finally the deviation of the prior kernel deviations is set to a little fraction of them. Otherwise speaking, $\sigma_{\sigma_0^{x(j)}}$ is set to a little fraction of $\sigma_0^{x(j)}$ for the j^{th} component of x.

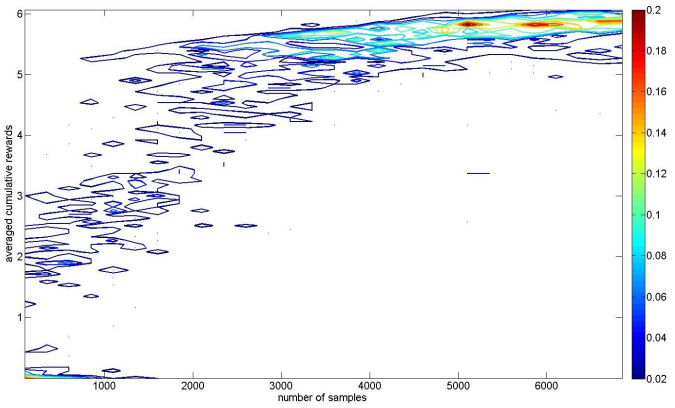


Fig. 8. Wet-chicken.

To sum up, the Gaussian prior on parameterization is chosen such that:

$$\begin{cases} \sigma_0^{x(j)} \propto x(j)_{\max} - x(j)_{\min} \\ \mu_{\alpha_0} = 0 \text{ and } \sigma_{\alpha_0} \propto \frac{r_{\max}}{1-\gamma} \\ \sigma_{\mu_0^{x(j)}} \propto \frac{x(j)_{\max} - x(j)_{\min}}{(n_s+n_a)\sqrt{p}} \\ \sigma_{\sigma_0^{x(j)}} \propto \sigma_0^{x(j)} \end{cases} \quad (61)$$

2) Wet-Chicken: In the wet-chicken problem (inspired by [24]), a canoeist has to paddle on a river until reaching a waterfall. It restarts if it falls down. Rewards increase linearly with the proximity of the waterfall, and drop off for falling. Turbulences make the transition probabilistic. More formally, the state space is $S = [0, 10]$ (10 being the waterfall position), the action space $A = [-1, 1]$ (continuously from full backward padding to full forward padding), the transition is $s' = s+a+c$ with $c \sim \mathcal{N}(0, \sigma_c)$, $\sigma_c = 0.3$ and the associated reward is equal to $r = \frac{s'}{10}$. If $s' \geq 10$ the canoeist falls, the associated reward is $r = -1$ and the episode ends.

To test the proposed framework, random transitions are uniformly sampled and used to feed the filter: at each time step k, a state s_k and an action a_k are uniformly sampled from $S \times A$, and used to generate a (random) transition to s'_k, with associated reward r_k, and the transition (s_k, a_k, s'_k, r_k) is the input of the algorithm. The results are shown on Figure 8. For each run of the algorithm and every 250 samples, the expected cumulative rewards for the current policy has been computed as an average of cumulative rewards over 1000 episodes which were randomly (uniform distribution) initiated (thus the average is done over starting states and stochasticity of transitions). Notice that the lifetime of the agent (the duration of an episode) was bounded to 100 interactions with its environment. Then a two dimensional histogram of those averaged cumulative rewards is computed over 100 different runs of the algorithm. In other words, the distribution of cumulative rewards over different runs of the algorithm as a function of the number of observed transitions is shown. The bar on the right shows the percentages associated to the histogram.

The optimal policy has an averaged cumulative rewards of 6. It can be seen on Figure 8 that the proposed algorithm can learn near optimal policies. After 1000 samples some of policies can achieve a score of 5 (84% of the optimal policy), which is achieved by a majority of the policies after 3000 samples. After 7000, very close to optimal policies were found in almost all runs of the algorithm (the mode of the associated distribution is at 5.85, that is 98% of the optimal policy). To represent the approximated Q-function, 7.7 ± 0.7 kernel functions were used, which is relatively few for such a problem (from a regression perspective).

Two remarks of interest have to be made on this benchmark. First, the observation noise is input-dependant, as it models the stochasticity of the MDP. Recall that here a constant observation noise has been chosen. Secondly, the noise can be far to Gaussianity. For example, in the proximity of the waterfall it is bimodal because of the shift of reward. Recall that the proposed filter assumes Gaussianity of noises. Thus it can be concluded that the proposed approach is quite robust, and that it achieves good performance considering that the observations were totally random (off-policy aspect).

3) Mountain Car: The second problem is the mountain-car task. A underpowered car has to go up a steep mountain road. The state is 2-dimensional (position and velocity) and continuous, and there are 3 actions (forward, backward and zero throttle). The problem full description is given in [11]. A null reward is given at each time step, and $r = 1$ is given when the agent reaches the goal.

A first problem is to find a parameterization for this task. The proposed one is adapted for continuous problems, not hybrid ones. But this approach can be easily extended to continuous states and discrete actions tasks. A simple solution consists in having a parameterization for each discrete action, that is a parametrization of the form $\theta = [\theta^{a_1}, \theta^{a_2}, \theta^{a_3}]$ and an associated Q-function $Q_\theta(s, a) = Q_{\theta^a}(s)$. But it can be noticed that for a fixed state and different actions the Q-values will be very close. In other words $Q^*(s, a_1)$, $Q^*(s, a_2)$ and $Q^*(s, a_3)$ will have similar shapes, as functions over the state space. Thus consider that the weights will be specific to each action, but the kernel centers and deviations will be shared over actions. More formally the parameter vector is

$$\theta = [(\alpha_i^{a_1})_{i=1}^p, (\alpha_i^{a_2})_{i=1}^p, (\alpha_i^{a_3})_{i=1}^p, (s_i^T)_{i=1}^p, ((\sigma_i^s)^T)_{i=1}^p]^T \quad (62)$$

the notation being the same as in the previous sections.

As for the wet-chicken problem, the filter has been fed with random transitions. Results are shown on Figure 9, which is a two-dimensional histogram similar to the previous one. The slight difference is that the performance measure is now the "cost-to-go" (the number of steps needed to reach the goal). It can be linked directly to the averaged cumulative rewards, however it is more meaningful here. For each run of the algorithm and every 250 samples, the expected cost-to-go for the current policy has been computed as an average of 1000 episodes which were randomly initiated (average is only done over starting states here, as transitions are deterministic). The

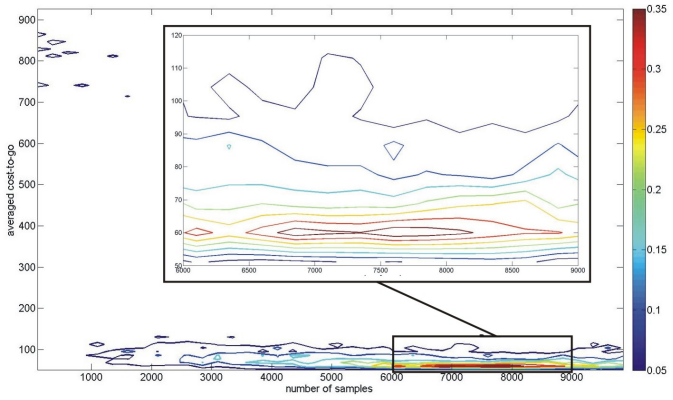


Fig. 9. Mountain car.

lifetime of the agent was bounded to 1000 interactions with its environment. The histogram is computed over 100 runs.

The optimal policy has an averaged cost-to-go of 55. It can be seen on Figure 9 that the proposed algorithm can find near optimal policies. After 1500 samples most of policies achieve a cost-to-go smaller than 120. After 6000 samples, policies very close to the optimal one were found in almost all runs of the algorithm (the mode of the associated distribution is at 60). To represent the approximated Q-function, 7.5 ± 0.8 kernel functions were used, which is relatively few for such a problem (from a regression perspective).

This problem is not stochastic, however informative rewards are very sparse (which can cause the Kalman gain to converge too quickly), transitions are nonlinear and rewards (that is observations) are binary. Despite this, the proposed filter exhibits good convergence. Once again it can be concluded that the proposed approach achieves good results considering the task at hand.

4) Comparison to Other Methods: For now the proposed algorithm treats the different control tasks as regression problems (learning from random transitions), thus it is ill comparable to state-of-the-art reinforcement learning algorithms which learns from trajectories. Nevertheless it is argued that the quality of learned policies is comparable to state-of-the art methods. Measuring this quality depends on the problem settings and on the measure of performance, however the Bayesian reward filter finds very close to optimal policies. See [24] for example.

In most approaches, the system is controlled while learning, or for batch methods observed samples come from a suboptimal policy. In the proposed experiments, totally random transitions are observed. However for the mountain-car problem it is often reported that at least a few hundreds of episodes are required to learn a near-optimal policy (see for example [11]), and each episode may contain from a few tens to hundreds steps (this depends on the quality of the current control). In the proposed approach a few thousands of steps have to be observed in order to obtain a near optimal policy. This is roughly the same order of magnitude for convergence speed.

VI. Conclusion

A Bayesian approach to online nonlinear kernel regression with a preprocessing sparse structure automatization procedure has been proposed. This method has proven to be effective (from a RMSE point of view) on a simple cardinale sine regression problem. This example demonstrated that the proposed approach compares favorably with the state-of-the-art methods and it illustrated how the uncertainty of generalization is quantified.

An approach allowing to regress a function of interest f from observations which are obtained through a nonlinear mapping of it has also been proposed as an extension. The regression is still online, kernel-based, nonlinear and Bayesian. This method has proven to be effective on an artificial problem and reaches good performance although much more nonlinearities are introduced.

Finally the proposed approach has been specialized into a general Bayesian filtering framework for reinforcement learning. By observing rewards (and associated transitions) the filter is able to infer a near-optimal policy (through the parameterized Q-function). It has been tested on two reinforcement learning benchmarks, each one exhibiting specific difficulties for the algorithm. This off-policy Bayesian reward filter has been shown to be efficient on these two continuous tasks.

However, this paper did not demonstrated all the potentialities of the proposed framework. The Bayesian filtering approach allows to derive uncertainty information over estimated Q-function which can be used to handle the so-called exploration-exploitation dilemma, in the spirit of [25] or [26]. This could allow to speed-up and to enhance learning. This uncertainty information can also be useful from a regression perspective if used for active learning. Moreover, the partial observability problem (the issue of non-directly observable state in RL) can be quite naturally embedded in such a Bayesian filtering framework, as the Q-function can be considered as a function over probability densities. Finally, the observation noise arising in the Bayesian Reward Filter is not purely white if the Markovian decision process has stochastic transition probabilities. The whiteness of this noise being a necessary assumption for the derivation of the sigma-point Kalman filter, which is a baseline of the proposed framework, this aspect should be investigated further.

Acknowledgment

Olivier Pietquin thanks the Région Lorraine and the European Community (CLASSiC project, FP7/2007-2013, grant agreement 216594) for financial support.

References

[1] M. Geist, O. Pietquin, and G. Fricout, "A Sparse Nonlinear Bayesian Online Kernel Regression," in *Proceedings of the Second IEEE International Conference on Advanced Engineering Computing and Applications in Sciences (AdvComp 2008)*, vol. I, Valencia (Spain), October 2008, pp. 199–204.

[2] C. M. Bishop, *Neural Networks for Pattern Recognition*. New York, NY, USA: Oxford University Press, 1995.

[3] B. Scholkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, MA, USA: MIT Press, 2001.

[4] V. N. Vapnik, *Statisical Learning Theory*. John Wiley & Sons, Inc., 1998.

[5] L. Feldkamp and G. Puskorius, "A signal processing framework based on dynamic neural networks with application to problems in adaptation, filtering, and classification," in *Proceedings of the IEEE*, vol. 86, no. 11, 1998, pp. 2259–2277.

[6] R. van der Merwe, "Sigma-Point Kalman Filters for Probabilistic Inference in Dynamic State-Space Models," Ph.D. dissertation, OGI School of Science & Engineering, Oregon Health & Science University, April 2004.

[7] C. M. Bishop and M. E. Tipping, "Bayesian Regression and Classification," in *Advances in Learning Theory: Methods, Models and Applications*, vol. 190. OS Press, NATO Science Series III: Computer and Systems Sciences, 2003, pp. 267–285.

[8] J. Vermaak, S. J. Godsill, and A. Doucet, "Sequential Bayesian Kernel Regression," in *Advances in Neural Information Processing Systems 16*. MIT Press, 2003.

[9] Z. Chen, "Bayesian Filtering : From Kalman Filters to Particle Filters, and Beyond," Adaptive Systems Lab, McMaster University, Tech. Rep., 2003.

[10] R. Bellman, *Dynamic Programming*, 6th ed. Dover Publications, 1957.

[11] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*, 3rd ed. The MIT Press, March 1998. [Online]. Available: http://www.cs.ualberta.ca/ sutton/book/the-book.html

[12] Y. Engel, S. Mannor, and R. Meir, "The kernel recursive least squares algorithm," *IEEE Transactions on Signal Processing*, vol. 52, pp. 2275–2285, 2004.

[13] M. Geist, O. Pietquin, and G. Fricout, "Online Bayesian Kernel Regression from Nonlinear Mapping of Observations," in *Proceedings of the 18th IEEE International Workshop on Machine Learning for Signal Processing (MLSP 2008)*, no. a53, Cancun (Mexico), October 2008, pp. 309–314.

[14] ——, "Bayesian Reward Filtering," in *Proceedings of the European Workshop on Reinforcement Learning (EWRL 2008)*, ser. Lecture Notes in Artificial Intelligence, S. G. et al., Ed. Lille (France): Springer Verlag, June 2008, vol. 5323, pp. 96–109.

[15] R. E. Kalman, "A New Approach to Linear Filtering and Prediction Problems," *Transactions of the ASME–Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.

[16] S. J. Julier and J. K. Uhlmann, "Unscented Filtering and Nonlinear Estimation," in *Proceedings of the IEEE*, vol. 92, no. 3, March 2004, pp. 401–422.

[17] D. Simon, *Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches*, 1st ed. Wiley & Sons, August 2006.

[18] A. L. Strehl, L. Li, and M. L. Littman, "Incremental model-based learners with formal learning-time guarantees," in *22nd Conference on Uncertainty in Artificial Intelligence*, 2006, pp. 485–493.

[19] I. Szita and A. Lőrincz, "Kalman Filter Control Embedded into the Reinforcement Learning Framework," *Neural Comput.*, vol. 16, no. 3, pp. 491–499, 2004.

[20] Y. Engel, "Algorithms and Representations for Reinforcement Learning," Ph.D. dissertation, Hebrew University, April 2005.

[21] C. W. Phua and R. Fitch, "Tracking Value Function Dynamics to Improve Reinforcement Learning with Piecewise Linear Function Approximation," in *ICML 07*, 2007.

[22] D. P. Bertsekas, *Dynamic Programming and Optimal Control*, 3rd ed. Athena Scientific, 1995.

[23] M. A. Carreira-Perpinan, "Mode-Finding for Mixtures of Gaussian Distributions," *IEEE Transsactions on Pattern Analalysis and Machine Intelligence*, vol. 22, no. 11, pp. 1318–1323, 2000.

[24] D. Schneegass, S. Udluft, and T. Martinetz, "Kernel Rewards Regression: an Information Efficient Batch Policy Iteration Approach," in *AIA'06: Proceedings of the 24th IASTED international conference on Artificial intelligence and applications*. Anaheim, CA, USA: ACTA Press, 2006, pp. 428–433.

[25] R. Dearden, N. Friedman, and S. J. Russell, "Bayesian Q-learning," in *Fifteenth National Conference on Artificial Intelligence*, 1998, pp. 761–768. [Online]. Available: http://www.cs.bham.ac.uk/ rwd/

[26] A. L. Strehl, L. Li, E. Wiewiora, J. Langford, and M. L. Littman, "PAC Model-Free Reinforcement Learning," in *23rd International Conference on Machine Learning (ICML 2006)*, Pittsburgh, PA, USA, 2006, pp. 881–888. [Online]. Available: http://paul.rutgers.edu/ strehl/

Examining Implementations of a Computationally Intensive Problem in GF(3)

Joey C. Libby
Faculty of Computer Science
University of New Brunswick
Fredericton, NB, Canada
g6x2d@unb.ca

Jonathan P. Lutes
Faculty of Computer Science
University of New Brunswick
Fredericton, NB, Canada
f9dz2@unb.ca

Kenneth B. Kent
Faculty of Computer Science
University of New Brunswick
Fredericton, NB, Canada
ken@unb.ca

ABSTRACT

Computing the irreducible and primitive polynomials under GF(3) is a computationally intensive task. A hardware implementation of this algorithm should prove to increase performance, reducing the time needed to perform the computation. Previous work explored the viability of a co-designed approach to this problem and this work continues addressing the problem by moving the entire algorithm into hardware. Handel-C was chosen as the hardware description language for this work due to its similarities with ANSI C used in the software implementation. A hardware design for the algorithm was developed and optimized using several different optimizations techniques before arriving at a final design.

Optimization; Handel-C; Galois Fields

1. INTRODUCTION

The performance of many software systems can be improved by the creation of custom hardware circuits that are capable of performing some or all of a software systems processing in a native hardware environment [8,9,10,11]. One major reason that software is implemented in hardware is the core features that a hardware implementation offers a system designer. The most important of these features is the inherent parallelism that is found in hardware systems such as Field Programmable Gate Arrays (FPGA).

The work presented in this paper is a continuation of work started in [1] and centers around the creation [14] of migrating a software system for the computation of irreducible and primitive polynomials over GF(3) completely to hardware, the issues that surrounded the migration and optimizations that were applied to the final hardware design using an automated parallelism extraction tool. The original work [1] concentrated only on implementing the computation intensive *multmod* function of the GF3 algorithm in hardware.

Further work presented includes further optimization of the design described in [14] and a discussion of several optimization techniques that were used to perform these optimizations.

This paper begins by presenting a brief background on some of the subject matter deemed relevant to the proper understanding of this paper. Following this the original research that lead to this project is presented, as well as work from another project on automated extraction of parallelism from Handel-C hardware definitions. The full hardware design is then presented followed by a discussion of the different optimization techniques used to optimize the design, as well as benchmarking results for each of the techniques.

2. BACKGROUND

This section will discuss the background information that is necessary for understanding this paper. This discussion includes Handel-C [2], Galois Fields [3] and the previous work that was completed.

2.1 HANDEL-C

The hardware implementation for this work was implemented in Handel-C [2]. Handel-C is a high level hardware description language that bears much resemblance to the ANSI C programming language. While Handel-C is very similar to ANSI C in many respects, there are some major differences between the two languages. Handel-C does not support the entire ANSI C specification. One of the more important features removed from Handel-C is support for runtime recursion. Handel-C, along with support for a subset of the ANSI C

specification, includes extra support for hardware descriptions. Included in this extended support are constructs for input and output, communications, and control flow constructs for controlling the parallelism of a design. Parallelism in a Handel-C program is defined by using the `par{}` and `seq{}` statement blocks. Sequential instructions wrapped in a `par{}` statement will be executed in parallel, while statements wrapped in a `seq{}` statement will be forced to execute sequentially. Example 1 shows `par` and `seq` statements in a simple Handel-C design.

The absence of runtime recursion support in Handel-C proved to be one of the more challenging aspects of this work. In most cases recursive algorithms can be easily converted to a non-recursive, loop based algorithm. This would prove to be problematic during the course of this work as several of the recursive functions written in the C algorithm proved to be resistant to conversion loops.

```
int 8 a,b,c,d,e,f,g,h;
a = 1; b = 2; c = 3; d = 4;
seq {
        d = a + b;
        e = c + d;
}
par {
       f = d+e;
       g = d*e;
}
```

Example 1: Example of `par` and `seq` Statements

2.2 GALOIS FIELDS AND THE ALGORITHM

A Galois Field is a finite order denoted by GF(p), where p is a prime or a power of primes [3]. A Galois Field of order p has only p elements, 0 though p-1. The focus of the algorithm implemented for this paper is Galois Fields of the order GF(3). These fields are of interest due to their application in pairing based cryptographic systems [4].

The C algorithm discussed in this paper describes the problem of enumerating all of the primitive and irreducible polynomials of a given order [5]. Irreducible polynomials are polynomials such that p(x) in F(x) is called irreducible over F if it is non-constant and cannot be represented as the product of two or more non-constant polynomials from F(x) [3]. A primitive polynomial is a polynomial such that F(X), with coefficients in GF(p) = Z/pZ, is a primitive polynomial if it has a root α in GF(pm) such that $\{0, 1, \alpha, \alpha^2, \alpha^3, \ldots, \alpha^{p^m-2}\}$ is the entire field GF(p^m), and moreover, $F(X)$ is the smallest degree polynomial having α as root [3].

The C algorithm consists of a number of functions that will now be detailed. Where applicable functions that are recursive are noted.

Add: Adds two polynomials under GF(3).
Subtract: Subtracts two polynomials under GF(3).
Mod: Takes the modulus of two polynomials under GF(3).
GCD: Find the greatest common divisor of two polynomials under GF(3) (recursive).
Multmod: Multiplies two polynomials under *mod* p.
Powmod: Finds the result of one polynomial raise to the power of another polynomial under GF(3).
Minpoly: Finds the minimum polynomial given a necklace.
Gen: Controls execution of the algorithm by cycling over all possible necklaces (recursive).

2.3 THE CO-DESIGNED SOLUTION

The previous implementation of the C algorithm did not attempt to migrate the entire software algorithm into a hardware system. Instead it was decided to explore a co-designed approach [1] where only a portion of the software would be translated into a hardware design and this hardware module would be called from the software running on a general purpose CPU.

2.3.1 THE CO-DESIGN

The first task was to determine how to partition the hardware and software for this project. Timing analysis of the original software showed that the multmod function, which computes multiplication between polynomials, is the most processing intensive portion of the software. It was found that 80% of the total processing time was spent in the multmod function. It was decided that a suitable solution to improve the performance of the system would be to implement the multmod function in hardware.

The design of the hardware was created based on careful analysis of the operations that are performed in the multmod function. These operations were then placed in a module called the Arithmetic Computation Unit (ACpU). Figure 1 shows a flowchart of the operations performed when executing a call to the multmod function.

The hardware partition also contains a module responsible for controlling the execution of calls to the multmod hardware. This module is called the Arithmetic Control Unit (ACtU). The ACtU is a finite state machine that is responsible for sequencing the operations that take place in the ACpU. The main requirement of this state

machine is to increase the performance of the hardware module by leveraging as much parallelism as is possible in the multmod operation. The design of the state machine for the ACtU can be seen in Figure 2.

Two different implementations of this co-designed hardware were then created. The first implementation was designed to be housed on an FPGA located on a board connected to its host computer via a PCI bus. Using this configuration software running on the host machine communicates with the hardware device via the PCI bus. One of the issues with this design, however, was communication delays between the hardware and software. In order to remove this communications gap, it was decided to explore another option using a System-on-chip architecture. In this design the FPGA is very closely integrated with an embedded processor. This allows the software partition to communicate with the custom hardware directly, without the use of the time costly PCI bus.

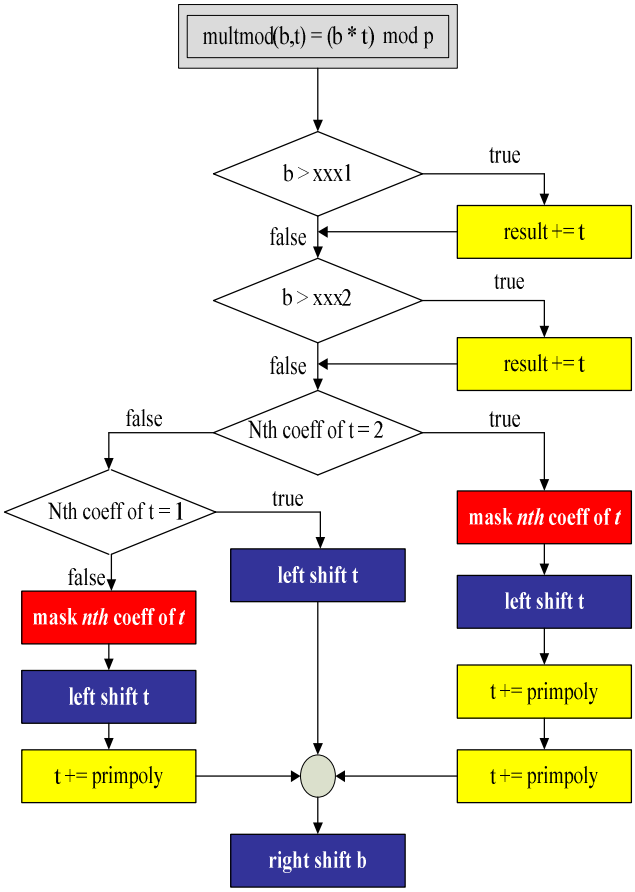


Figure 1: Arithmetic Computation Unit

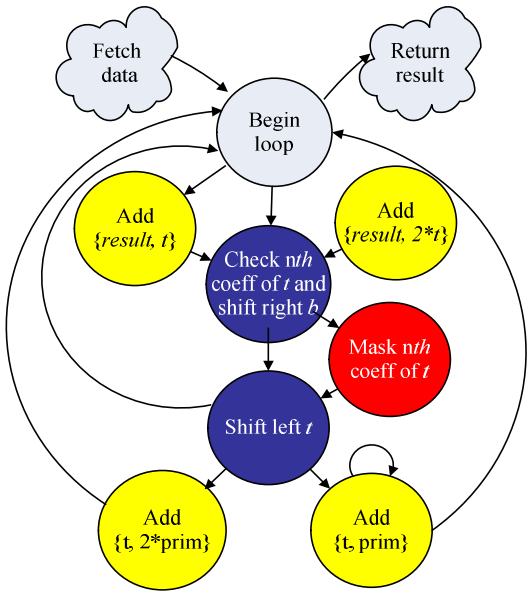


Figure 2: Arithmetic Control Unit

2.3.2 THE PCI-BASED SOLUTION

For the PCI-based solution the reconfigurable hardware receives and sends data to the software partition over a PCI bus. For this implementation the software was running on a Windows 2000 PC with a 1.8 Ghz Intel Xeon processor with 1 GB of RAM. The FPGA development board used for this project is an APEX PCI Development board with an Altera Apex 20KC1000CF672 FPGA [15] supporting 32 and 64 bit PCI communications at 33 and 66 Mhz [16]. The software running on the host machine is a modified version of the original software implementation. The modifications allow the software to call the hardware when necessary for completing computations. A high level view of the structure of this loosely coupled co-designed system can be seen in Figure 3.

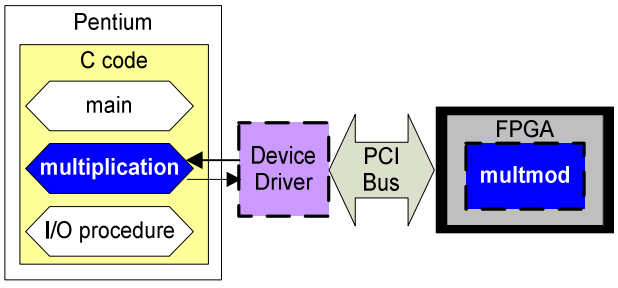


Figure 3: Overview of loosely coupled system

In order to interface the software partition with the hardware device it was necessary to develop a PCI device driver for the system [17].

The modifications to the original C code allowed all calls made to the multmod function to be redirected to the hardware device. In order to do this, input data for the calls to the multmod function are redirected to the PCI device driver. The device driver is then responsible for initiating communications with the FPGA and sending all required data for the call. In this implementation the configuration is static and must be downloaded to the FPGA before runtime. This configuration contains four modules: the ACpU, ACtu, PCI Core and PCI control unit.

The hardware implementation of the *multmod* algorithm was implemented in Verilog and was targeted to an Amirix AP1000 development board [6]. This development board was chosen as the target platform because of its on-chip PowerPC processor that is directly connected to the FPGA fabric. This feature allowed the software to be executed on a platform that is more tightly coupled with the FPGA and removed the need to create a PCI bus driver for the work. Figure 4 shows an overview of the co-designed system and Table 1 shows the benchmarking results for this implementation.

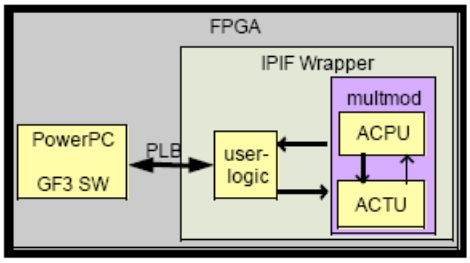


Figure 4: Co-Designed System Overview[1]

While a performance increase was realized by moving to the co-designed system, it was found that several factors severely limited the overall performance of the system. The slow speed of the embedded processor running the software portion of the system was one issue that arose. The 200mhz clock speed of this processor simply was not fast enough to hold pace with the faster general purpose processors that would normally run the full software implementation [1].

Also a major problem, more so than the slow clock rate of the embedded processor running the software portion of the system, is the communications between the hardware and the software system. Communications prove to be the Achilles heel of this work, as well as many other co-design works [7]. The amount of data communications that is necessary between the hardware and the software is so great that it limits the maximum throughput of the system, which has a huge impact on performance.

The only solution to this problem is to move the entire system into hardware, completely eliminating the communication channels. This will allow the system to operate at full speed, only having to access communication channels when retrieving jobs and reporting results.

2.4 AUTOMATED EXTRACTION OF PARALLELISM

Identification of simple parallelism, that is sequential blocks of hardware code that can be executed in parallel, can have a huge impact on the performance of the hardware system that is being designed. The tool that will be used to apply optimizations for the purpose of this paper can be found in [12].

Given a HandelC source file, this tool is capable of parsing and extracting simple parallelism from the source file. This information is then relayed to the hardware designer who can implement the proposed changes in order to build a more optimized version of the original hardware design.

Figure 5 shows an overview of the operation of the automated parallelism extraction tool. The tool operates by taking, as input, a HandelC hardware definition file provided by a software programmer or hardware designer. From this source file the tool creates an abstract syntax tree, annotated with additional information that is required to compute the dependency graph from the source file. Upon completion of the syntax tree, it is used to generate a dependency graph structure for the hardware design. This dependency graph structure is then used to determine which individual lines of source can potentially be executed in parallel. Currently the tool then applies a greedy algorithm which builds as large and as many parallel blocks as possible from the remaining available lines of source code. This approach generates large parallel blocks which in turn reduce the overall run time of computations on the hardware.

Once the tool has determined where parallel blocks may be added to the source hardware design, it produces a report for the developer that details the necessary modifications that must be performed in order to exploit the available parallelism. Table 2 illustrates the report output of the tool. After potentially several iterations with the tool, the developer can then input their HandelC specification into the typical tool flow starting with the Agility DK tool suite [2].

Degree	SFW @1.8GHz (sec)	Altera @66MHz (sec)	%	Xilinx @80MHz (sec)	%
2	0.00004	0.00002	*37.6*	0.00001	*24.8*
3	0.00018	0.00009	*46.6*	0.000056	*30.7*
4	0.00076	0.00034	*44.2*	0.000228	*30.0*
5	0.00298	0.00145	*48.6*	0.001005	*33.7*
6	0.01495	0.00519	*34.7*	0.003664	*24.5*
7	0.04043	0.02005	*49.6*	0.01439	*35.6*
8	0.14300	0.07123	*49.8*	0.051788	*36.2*
9	0.54600	0.25595	*46.9*	0.188174	*34.5*
10	1.89800	0.89412	*47.1*	0.663513	*35.0*
11	6.24000	3.08083	*49.4*	2.302203	*36.9*
12	22.30800	10.58020	*47.4*	7.963267	*35.7*
13	74.78900	35.12896	*47.0*	26.53804	*35.5*
14	263.53600	120.09697	*45.6*	91.323996	*34.7*
15	888.30300	400.77343	*45.1*	306.075094	*34.5*
16	2985.50200	1343.56091	*45.0*	1049.41517	*35.9*
17	10192.85900	4424.87400	*43.4*	n/a	n/a
18	32658.34090	14642.10675	*44.8*	n/a	n/a

Table 1: Co-designed Performance Results [1]

In testing, this tool has proven that it is capable of finding, on average, 78% of the simple straight line parallelism that exists in a hardware design. Tables 3 and 4 shows manual and automatic optimization benchmark results for AES encryption and LZ77 decryption hardware circuits optimized using this tool. [12]

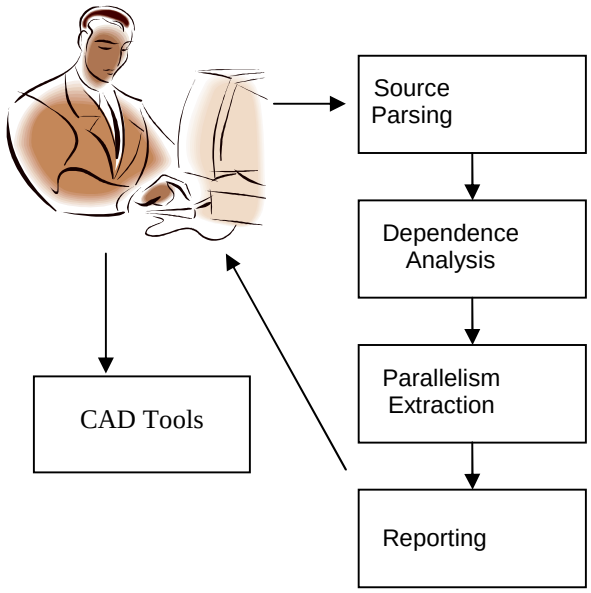


Figure 5: Automated Parallelism Extraction

Source Line#	Source	Action
*	Par {	Add
1	Statement 1	None
2	Statement 2	None
3	Statement 3	None
7	Statement 7	Move -
8	Statement 8	Move –
*	}	Add
4	While	Move +
5	Statement 5	Move +
6	Statement 6	Move +
9	Statement 9	Move +

Table 2: Tool Report [12]

Test	Par Blocks
LZ77	5
AES	13

Table 3: Manual Benchmark Statistics [12]

Test	Par Blocks Found	% of Total
LZ77	4	80%
AES	10	76%

Table 4: Automated Benchmark Statistics [12]

3. THE HARDWARE SOLUTION

In order to alleviate the performance degradation caused by communications between the hardware and software in the co-designed system, as well as the low performance of the general purpose processor, a full implementation was created in hardware. This implementation was written in Handel-C which allowed the hardware implementation to very closely mimic the software algorithm wherever possible.

Much of the ANSI C code that was created for the algorithm was capable of being directly translated into Handel-C. The code that was directly translated required only minimal modification to make it compatible with the Handel-C language. Some of these changes included re-definition of storage elements such as arrays to use static sizes instead of being dynamically allocated. Another trivial modification that was required in several places was the un-nesting of function calls. Handel-C does not

support the usage of nested function calls of the form `foo( bar(x,y),z)`. This necessitated rewriting some C code to call these functions sequentially using temporary variables to store the return value of the nested function call.

3.1 REMOVING RECUSION FROM THE ALGORITHM

Once the code was converted to Handel-C syntax all that remained was removing the recursion that exists in several of the functions in the software. The functions that required modification to remove recursion were the Gen and GCD functions. Both functions were translated to their loop based variants. Example 1 shows how the recursive function definition for the GCD was transformed into a loop.

```
Poly_GF3 gcd(Poly_GF3 a, Poly_GF3 b){
  if(!b.top && !b.bot) return a;
  return gcd(b, mod(a, b));
}
```

Example 1 (a): Recursive GCD Definition

Once the recursion was removed from the software functions they were implemented in Handel-C. Following the implementation in Handel-C, each function required verification to ensure that the hardware versions were equivalent to their software counterparts.

```
Poly_GF3 gcdx(Poly_GF3 a, Poly_GF3 b )
{
  Poly_GF3 c,zero;

  zero = {0,0};
  while (a.top || a.bot)
  {
     c = a;
     modx(b,a);
     a = modxResult;
     b = c;
  }
  return b;
}
```

Example 1 (b): Non Recursive GCD Definition

One of the larger challenges for this project was the removal of recursion from the Gen function. Example 2 shows the Gen function with a manual stack.

```
inline void Push(int *pos, int t, int
p, int j, int s) {
  stack[*pos][0] = t;
  stack[*pos][1] = p;
  stack[*pos][2] = j;
  stack[*pos][3] = s;
  (*pos)++
}

void Gen(unsigned t, unsigned p) {
  unsigned int j, top, state;

  top = 1;
  Push(top, 1, 1, 0, 0);
  top--;
  while (top > 0) {
    top--;
    t = stack[top][0];
    p = stack[top][1];
    j = stack[top][2];
    state = stack[top][3];
    if (t > N) {
      if (p == N) CheckIt();
    }
    else {
      switch (state) {
        case 0: {
          a[t] = a[t-p];
          Push(top, t, p, 0, 1);
          Push(top, t+1, p, 0, 0);
          break;
        }
        case 1: {
          j = a[t-p] + 1;
          if (j <= 2) {
            a[t] = j;
            Push(top, t, p, j, 2);
            Push(top, t+1, t, 0, 0);
          }
          break;
        }
        case 2: {
          j++;
          if (j <= 2) {
```

```
            a[t] = j;
            Push(top, t, p, j, 2);
            Push(top, t+1, t, 0, 0);
          }
          break;
        }
        default: { break; }
      }
    }
  }
}
```

Example 2: Non Recursive Gen Function

In order to remove recursion from the Gen function to make it compatible with hardware it was necessary to express the recursion in the form of a loop. One of the issues with recursion in hardware is the requirement for virtually unlimited memory within which the recursion can grow. It was found through careful analysis that the depth of the recursive calls to the Gen function was bounded by the input value N. Because of this upper bound it became possible to express the recursion as a loop by creating a stack on which the state of each "recursive call" would be saved.

Each iteration of the loop would emulate a recursive call to the Gen function, pushing the previous state onto the stack and then beginning computation on the new state. Once computation is finished in the final recursive call, the result is computed by tracing back through the states stored on the stack to compile the final result.

This method provides a means to implement the Gen function in hardware, but it does have some limitations. In order for this method to work it is necessary to have a bound on the size of the computation. In this case the input value N is known at compile time, allowing the size of the stack to be sufficiently large. In some cases, however, the input size may also be limited by the available hardware resources, as a very large input value may cause the stack to consume all available resources, or more resources than are actually available for the design. For this particular application, the stack size grows to a rather small size, N, where the algorithm is exponentially bound by the same N. Thus, for this application we can easily fit the stack in the dedicated memory bits that are available on the FPGA device.

3.2 HARDWARE VERIFICATION

In order to verify that the hardware functions, especially the functions that were transformed from recursive to non-recursive, behaved as intended it was necessary to perform some verification tests. Test cases included boundary cases as well as a large number of randomly generated inputs to the functions.

Verification of the transformed recursive functions was performed in two stages. In the first stage, the non-recursive algorithm was tested as a software algorithm. Test cases were run against both the recursive and non-recursive versions of the functions and their return values were compared. Following running the test cases on both the recursive and non-recursive functions it was deemed that the recursive and non-recursive functions were both functionally equivalent and so passed verification.

Verification of the Handel-C hardware code was slightly more involved than testing software code against software code. The Handel-C hardware code was again tested using the same set of test cases used for testing the recursive functions. These test cases were first ran in the software version of the system, recording the results for each test. The same tests were then performed on each hardware function individually, running the hardware in a simulation environment. The results were also recorded and compared to those produced by the software for the same tests.

Following verification of the hardware definition it was deemed that the hardware definition is equivalent to the software algorithm so the work could proceed to benchmarking.

4. BENCHMARKING

In order to benchmark the hardware design of the GF(3) algorithm, it was necessary to synthesize the hardware definition to produce a hardware programming file. It was decided that the hardware would not be programmed onto a physical device for testing, but tests would be performed in a simulation environment in order to facilitate the gathering of statistics.

The Handel-C definition was first compiled using the Agility Handel-C compiler to produce both an executable simulation file as well as a synthesizable VHDL description file [2]. The execution simulation kernel was used to gather timing results for the hardware system and the VHDL description file was used to gather resource usage and clock speed statistics. Resource usage and clock speed statistics were gathered by synthesizing the VHDL specification in Xilinx ISE [18] targeting a Virtex II FPGA (XC2VP100). This FPGA is the same device used for gathering the results for the co-designed GF(3)

algorithm. The results in Table 5 show the resource usage and clock frequency for the design.

Clock Speed	Slices	Flip flops
68.523	23952	14579

Table 5: Resource Usage and Clock Frequency

Runtimes for the hardware were gathered by running the simulation kernel on different degrees ranging from 3 to 12. Cycle statistics were gathered for each run, and using the clock rate gathered from the Xilinx synthesis tool a run time was calculated. These run times are compared to the runtimes of the software in Table 6. Software run times were gathered on a 1.8 Ghz Pentium .

N	Cycles	HW Time (Seconds)	SW Time (Seconds)
3	15158	0.000212	0.061
6	899241	0.0131	0.075
8	13052272	0.1905	0.241
10	170959343	2.4949	2.102
12	2072543280	30.2495	26.603

Table 6: Runtime Comparison

On inspection of the results, it can be clearly seen that the hardware version of the algorithm, in its current form, does not surpass the performance of the software algorithm. While the hardware algorithm does not perform better than the software, the performance gap between the two is negligible when taking into account the speed grade difference between the hardware running at 68.523 Mhz and the software running on a 2.8 Ghz processor.

Taking this into account it was decided to attempt to improve the hardware design further by attempting to optimize the design for a hardware environment. Until this point the software had been converted to a hardware definition almost verbatim, ignoring any of the traditional hardware specific features such as parallelism.

5. OPTIMIZATION

The optimization that was chosen for this design was the addition of parallelism. The software design did not take into account any of the areas of parallelism that might lead to greater performance for the hardware system. For the purpose of this work, only simple optimizations were attempted. Individual statements that were capable of parallel execution were grouped into parallel blocks using the Handel-C `par` construct.

The parallel blocks were identified using a combination of both an automated parallelism detection tool [12] as well as manual optimization. This tool allows for the automatic identification of code that can potentially be executed in parallel. Currently the tool does not modify the Handel-C source directly and requires intervention from the designer to take advantage of code that is identified as parallel. The automated tool found a large portion of the available parallel blocks, and then manual code inspection was used to find more parallel blocks that the tool was unable to identify.

Clock Speed	Slices	Flip flops
68.909	23603	14383

Table 7: Resource Usage and Clock Frequency

The design was then simulated to gather clock cycle statistics for running the design at several different input values. These clock cycles were used, along with the clock rate statistic from Table 7 to generate the final runtime statistics for the new hardware which can be found in Table 8.

N	Cycles	Percent Reduction	HW Time (Secs)	SW Time (Secs)
3	10916	27.9%	0.000158	0.061
6	581396	42.4%	0.00843	0.075
8	8319569	36.3%	0.1207	0.241
10	108497030	36.5%	1.5745	2.102
12	1312560988	36.7%	19.0477	26.603

Table 8: Parallel Runtime Comparison

In order to emphasize the impact of using automated software for identifying parallelism optimizations in this hardware design it should be noted that the automated optimization process took less than a second to complete. Even in this case where it was found that several parallel blocks that were identified did not compile correctly the amount of time saved from doing a completely manual optimization is quite high. It was found in [13] that manually optimizing this design, with no previous knowledge of the available parallel blocks takes a skilled hardware designer approximately 8 hours of testing and refining. Even taking into account approximately one hour of testing to identify the two parallel blocks that did not behave properly after compilation, a time savings of approximately 7 hours was achieved.

After automated optimization of the hardware algorithm was complete, a brief manual inspection of the

remaining code was performed. This inspection yielded 8 more parallel blocks that were not found by the automated tool for a total of 32 `par` blocks of two or more sequential statements. This additional manual optimization took approximately two hours to complete. Parallel execution statements (`par{}`) were added to the design and the design was recompiled, again producing both a simulation kernel and a VHDL definition file for hardware synthesis. Table 9 shows the synthesis results gathered from the Xilinx ISE, again targeting the Virtex II FPGA (XC2VP100).

Clock Speed	Slices	Flip flops
68.813	23348	14245

Table 9: Resource Usage and Clock Frequency

Using the clock speed from Table 9 and the cycle statistics gathered from the simulation kernel the runtime statistics for the hardware algorithm can be calculated. Table 10 shows the new runtime statistics for the parallel hardware design. Also shown in Table 10 is the percentage reduction of clock cycles between the original non-parallel design and the parallel design.

Table 9 shows that a small increase, 0.290 Mhz, in clock speed was realized when moving from the non-parallel to the parallel design. The number of slices and flip flops utilized by the design was also reduced slightly. This is explained by the manner in which the handelC specification is synthesized. During synthesis the handelC code is transformed into a datapath and a finite state machine controller. By using `par{}` statements, states within the FSM controller are merged, thus reducing the size of the circuit. Contrary to some thoughts, the par{} statement does not add redundancy, multiple computational units, unless a shared function is used. Thus, exploiting parallelism typically gives the benefit of both faster computation and a smaller circuit. Figure 6 shows a comparison of the parallel and non-parallel hardware against the software implementation.

N	Cycles	Percent Reduction	HW Time (Secs)	SW Time (Secs)
3	8621	43.1%	0.000125	0.061
6	548189	39.0%	0.0079	0.075
8	8089562	38.0%	0.1176	0.241
10	106849548	37.5%	1.5527	2.102
12	1352768511	34.7%	19.6586	26.603

Table 10: Parallel Runtime Comparison

It can be seen in Figure 6 that the parallel version of the hardware outperforms the software implementation of the algorithm at all data points gathered for this work. It also appears that the hardware will continue to outperform the software even when computing orders higher than 12. Figure 7 illustrates the trend in the percentage difference between the hardware and software algorithms. This figure clearly shows that the rate of convergence between the hardware and software run times is slowing and that the hardware will continue to outperform the software.

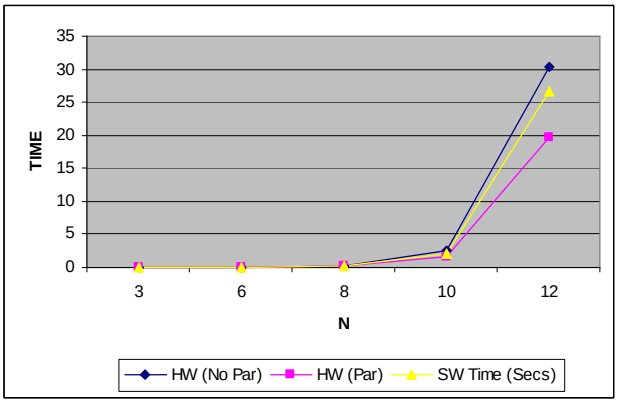


Figure 6: Results Comparison

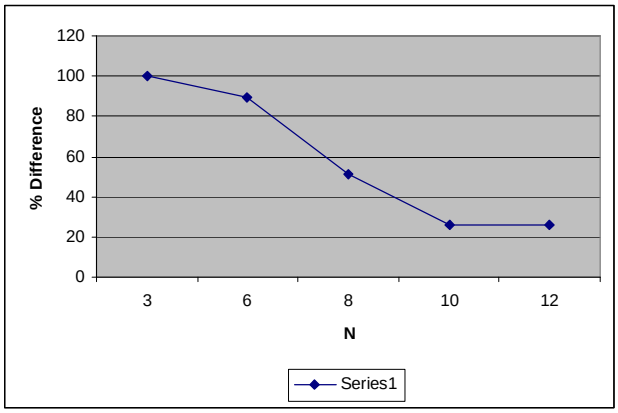


Figure 7: Execution Time Percentage Difference Between Parallel Hardware and Software

5.1 Further Optimizations

After completion of benchmarking of the parallelized design it was decided that an attempt would be made to identify any other optimizations that were possible in the design. After careful analysis of the design source code it was determined that there were two such types of optimizations that were suitable for this design.

The first optimization that was performed modified the method used to return values from function calls. In the original version of the hardware design, returns from non-recursive functions were stored in global variables,

and then assigned to the local variables as a result upon returning from the function call. This was done initially to reduce the amount of logic that was required in order to allow some functions to be called in parallel. Performing this modification entailed simply rewriting the function to return a value instead of writing it into a global variable.

The second optimization that was performed was also a side effect of applying the return value optimization. Changing the return from a global variable to an actual return value caused many function calls that could possibly be made in parallel to break. In order to fix this all of these functions were changed to inline functions. What this means is that instead of the compiler synthesizing control logic for a single logic core for each function, it creates a new instance of the logic for each function every time that function is called. This optimization has the effect of increasing the resource usage of the design, but decreasing the overall runtime of the design by reducing the amount of steps required to perform a function call.

Upon completion of these new optimizations the design was once again benchmarked, and the resource usage and clock rate statistics shown in Table 11 were gathered.

Clock Speed	Slices	Flip flops
58.998 Mhz	35991	34945

Table 11: Resource Usage and Clock Frequency

A simulation of the newly optimized hardware design was then performed, and cycle statistics for several input values were gathered. These cycle statistics, in conjunction with the clock rate given in Table 11 provide the actual runtimes shown in Table 12.

N	Cycles	Percent Reduction	HW Time (Secs)	SW Time (Secs)
3	7339	14.87	0.000124	0.061
6	464256	15.31	0.00786	0.075
8	6776626	16.23	0.11486	0.241
10	88118822	17.53	1.49356	2.102
12	1089884747	19.40	18.4732	26.603

Table 12: Parallel Runtime Comparison

As can be seen in Table 12, the newly optimized version of the hardware does indeed increase the performance of the design by an average factor of 1% over the parallelism only design.

Table 11 shows a comparison of the hardware run time of all four version of the hardware. This table illustrates the differences in performance gains along with the large amount of performance that was gained through the final round of optimizations.

N	Original	Automated Parallelism	Manual and Automated Parallelism	Final
3	0.000212	0.000158	0.000125	0.000124
6	0.0131	0.00843	0.0079	0.00786
8	0.1905	0.1207	0.1176	0.11486
10	2.4949	1.5745	1.5527	1.49356
12	30.2495	19.0477	19.6586	18.4732

Table 13: Comparison of all Designs

5.2 Comparison of Multmod Software and Hardware

This section will show a comparison of the hardware implemented for the Multmod function against the original software implementation. This section is meant to support that findings of this paper by showing that not only does the entire system outperform the software implementation of the GF(3) software, but also outperforms the implementation of the Multmod function which was implemented in hardware originally in [1].

Figure 8 shows the results that were gathered for these benchmarks. These statistics were gathered by adding cycle accurate counters to the final optimized version of the Multmod function, and then rerunning all of the simulations used previously. Statistics from these cycle counters were then gathered.

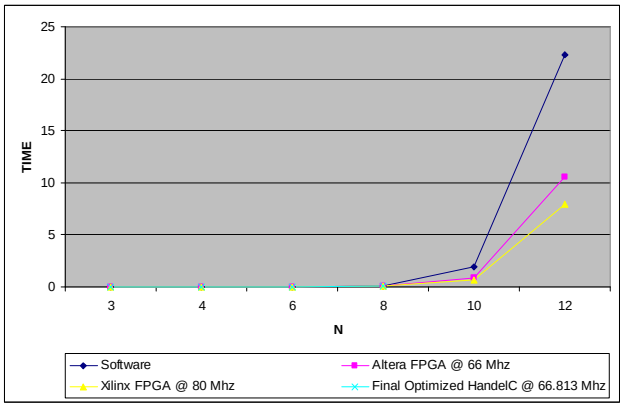


Figure 8: Multmod Comparison

Figure 8 clearly shows that while the optimized HandelC design runs at a slower clock rate then the Xilinx implementation, it still outperforms this implementation. This highlights the speed improvements that can be achieved by applying specific optimizations to a hardware design, as well as the speed improvements from moving from a co-designed environment to a full hardware implementation.

6. Conclusion

Based on the results gathered after optimizing the Handel-C design for the GF(3) primitive and irreducible polynomials algorithm it can be said that this work is a success. The entire algorithm was implemented in hardware and verified to function correctly. The results found in Section 5 highlight the performance of the hardware system, which outperforms the software on all test points up to order 12. It also appears that, based on Figure 2, the software will continue to outperform the hardware on higher orders. Figure 9 shows a comparison of the final results gathered. This figure clearly shows that the final optimized version of the hardware outperforms the original software version at all collected data points.

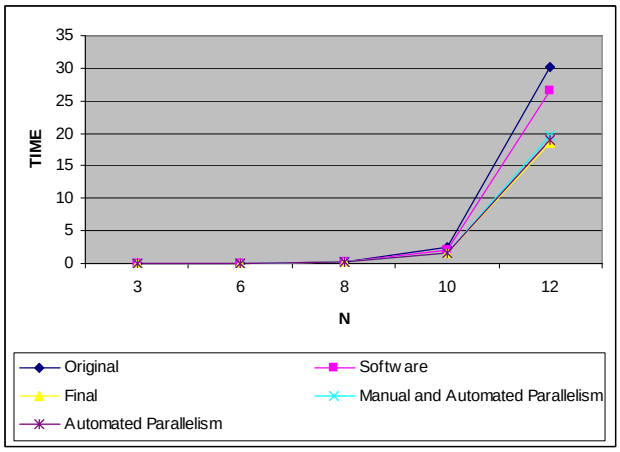


Figure 9: Final Results Comparison

Figure 10 highlights the amount of user effort that was required to complete each stage of the optimization process. The figure displays improvements based on the original, un-optimized version of the hardware, based on percentage increase in performance per minute of development time to achieve that performance increase.

Finally Figure 11 shows the increase in logic usage between the original and optimized versions of the hardware. This figure highlights how, while performance has increased, in the case of the final optimized version so have the resource requirements to implement the design in hardware.

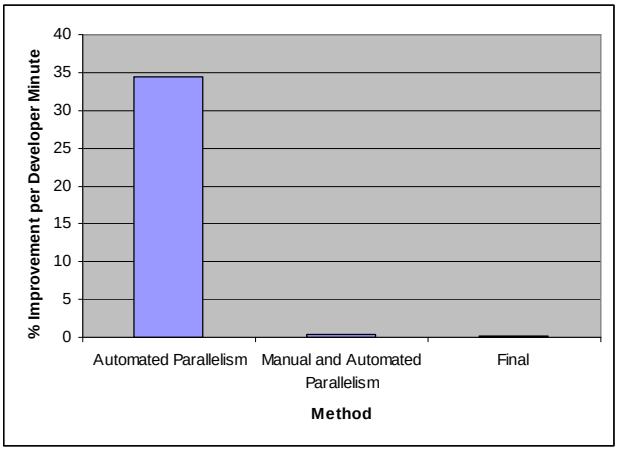


Figure 10: Developer Time Per Percentage Performance Increase

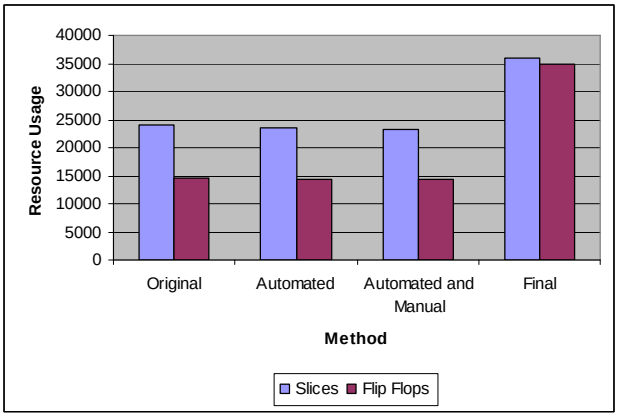


Figure 11: Resource Usage Comparison

7. Future Work

While the work can be considered a success, there is still much work to be done to further improve the performance of the system. At present only simple parallelism has been identified in the system. While parallelism between individual statements in a Handel-C program can greatly increase performance, there can be even greater performance gains from exploiting loop based parallelism or parallelism between different functional units.

Another optimization that may greatly benefit this work is the identification and implementation of a pipelined data path. A pipelined data path may increase the throughput of the algorithm by increasing the amount of work that is done per clock cycle by breaking the algorithm down into functional units that can operate in parallel much like an assembly line.

References

[1] K. Kent, B. Iaderoza, and M. Serra. Codesign of a Computationally Intensive Problem in GF(3), International Workshop on Rapid System Prototyping, pp. 10-16, May 2007.

[2] Agility Design Solutions, Handel-C Reference Manual, Website: www.agilityds.com. Accessed: May 2008

[3] G. Birkhoff, and S. Mac Lane. A Survey of Modern Algebra, 5th ed. New York: Macmillan, 1996.

[4] D. Page and N. P. Smart, "Hardware Implementation of Finite Fields of Characteristic Three". Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems, pp. 529-539, 2002.

[5] G. Lee, and F. Ruskey, Listing All Irreducible and Primitive Polynomials in GF(3),. Technical Report (University of Victoria, Canada), unpublished, 2006.

[6] AP1000 FPGA Development Board User Guide. User Guide Manual Version 2. AMIRIX Systems Inc, Halifax, Nova Scotia, Canada. 2005.

[7] M. Moazeni, A. Vahdatpour, K. Gururaj, and M. Sarrafzadeh, Communication Bottleneck in Hardware-Software Partitioning. In Proceedings of the 16th International ACM/SIGDA Symposium on Field Programmable Gate Arrays, pp. 262, 2008.

[8] R. Andraka, A Survey of CORDIC Algorithms for FPGA based Computers, 1998 ACM/SIGDA 6th International Symposium on Field Programmable Gate Arrays, pp. 191-200, 1998.

[9] M. Mylona, D. Holding, and K. Blow, DES Developed in Handel-C, London Communications Symposium, 2002.

[10] M. Serra, and K. B. Kent, Using FPGAs to Solve the Hamiltonian Cycle Problem, International Symposium on Circuits and Systems, pp. 228-231, vol. III, May 24-28, 2003.

[11] T. G. Noll, Application Specific eFPGAs for SoC Platforms, 2005 IEEE VLSI-TSA International Symposium on VLSI Design, Automation and Test. April 2005.

[12] J. C. Libby, and K. B. Kent, Automatic Identification of Concurrency in Handel-C, International Symposium on Digital Systems Design, pp. 660-664, August 2008.

[13] J. C. Libby, and K. B. Kent, A Methodology for Rapid Optimization of HandelC Specifications, submitted to the 20th IEEE/IFIP International Symposium on Rapid System Prototyping, June 2009.

[14] J. C. Libby, J. P. Lutes, and K. B. Kent, A Handel-C Implementation of a Computationally Intensive Problem in GF(3), International Conference on Advances in Electronics and Micro-electronics, ENICS 2008, pp. 36-41, October 2008.

[15] APEX 20KC Programmable Logic Device. *Data Sheet Version 2.2*., Altera Corporation, 2002.

[16] APEX PCI Development Board Data Sheet. *Data Sheet Version 2.1*. Altera Corporation, 2002.

[17] J. Hannula, “Is High-Level Design Representation Worthwhile?”, *M.Sc thesis*, University of Victoria, Canada, 2004.

[18] Xilinx Incorporated, Website: http://www.xilinx.com/ , Accessed: 1/29/2009.

How to compare and exploit different techniques for unit-test generation

Alberto Bacchelli
bacchelli@cs.unibo.it

Paolo Ciancarini
ciancarini@cs.unibo.it

Davide Rossi
rossi@cs.unibo.it

Department of Computer Science
University of Bologna, Italy

Abstract

The size and complexity of software is continuously growing, and testing is one of the most important strategies for improving software reliability, quality, and design. Unit testing, in particular, forms the foundation of the testing process and it is effectively supported by automated testing frameworks. Manual unit-test creation is difficult, monotonous and time-consuming. In order to reduce the effort spent on this task, several tools have been developed. Many of them can almost automatically produce unit tests for regression avoidance or failure detection.

This paper presents a practical comparison methodology to analyze different unit-testing creation tools and techniques. It validates the effectiveness of tools and spots their weaknesses and strengths. The validity of this methodology is confirmed through a real case experiment, in which both the manual implementation and different automatic test generation tools (based on random testing) are used. In addition, in order to integrate and exploit the benefits of each technique, which result from the comparison process, a testing procedure based on "best practices" is developed.

Keywords: testing; comparison methodology; failure detection; regression testing; automatic test generation tools

1 Introduction

Testing is an important and widely-accepted activity to verify a software at runtime. It can be performed at three different stages with increasing granularity: single class (or module) testing, classes (or modules) group testing, whole system testing [35]; which are usually referred to as: unit testing, integration testing and system testing. Each stage has its own difficulties and strategies, and different techniques are available. In this paper, we decided to concentrate our efforts on unit testing, which constitutes the foundation for other testing levels. In particular, we consider object-oriented software and the Java programming language, but our results could be easily transposed to be useful for different programming languages and paradigms.

The *x*Unit testing [2] framework was created in order to improve object-oriented programmer productivity in developing and running unit-test cases. Through it, it is possible to easily write unit tests that exercise and make assertions about the code under test. Each test is independent of each other and it is usually written in the same language as the code they test. The *x*Unit framework is intentionally simple to learn and to use, and this was a key design criteria: the authors wanted to be sure that it "was so easy to learn and to use that programmers would actually use it" [28]. The developer, with a simple command, could automatically run all the *x*Unit tests he created and then, after the execution, receive the report generated by the framework with the number of successes and the details of any failures.

In this article, we consider JUnit [1], which is the Java flavor of the *x*Unit framework. Although it is the "de-facto" standard tool to automatically execute Java unit tests, there are other interesting alternatives that could be effectively used (e.g., [10]).

In industry, testing is used for quality assurance, because it provides a realistic feedback on system behavior, reliability and effectiveness. At the same time, testing -especially unit testing- can be tedious and difficult, and it often consumes a considerable amount of time [31]. For this reason, research is now highly active in producing testing techniques capable of automatically creating unit tests. The usage of these tools is not already equally standardized as JUnit usage.

In a previous article [7], we compared the effectiveness of unit-test cases which were automatically created with manually written ones. We used various automatic test creation tools, based on random testing, and we proved that they can produce trustworthy and useful test cases, and can improve and speed-up testing engineers' tasks. We also showed the advantages which result from the manual creation of unit tests, that cannot yet be achieved by automatic tools. Finally, we noted that the success rate of this kind of testing depended to a great degree on proper planning and proper use of these testing tools. Thus, we briefly outlined

some "best practices" for integrating the manual and automated test processes, in order to effectively produce unit-test cases and obtain benefits from all the techniques.

As scientific research is highly active and motivated in this particular field, the tools we examined and compared in [7] have since been further enhanced [33], studied in more detail [32] or replaced with more effective tools [15, 39]. Also the only commercial software we included [5], which was highly experimental, was replaced with a more stable version [4] in order to address the company's business.

These automatic unit-test generation tools evolve extremely quickly, and thus we consider it significant to outline a comparison methodology to help the test engineer to analyze the capabilities of different tools, which can thus be exploited and fit into the correct testing "best-pratices".

In order to effectively extract this methodology, we use the real-world case study we conducted in [7]. We show, in detail, the comparison method we used and its validity. Then, we see what its advantages and shortcomings are, and how it could be usefully extended. Finally, we outline how the results obtained could be used in "best practices" capable of exploiting and integrating the different techniques which were analyzed.

In Section 2 we introduce unit testing and we show its main objectives. Then, in Section 3, we present the experimental study showing the environment in which we conducted it, describing which unit testing techniques and tools we used, and discussing its validity. Later (Section 4), we point out the comparison we adopted to study this real world case and we explain how we analyzed the different facets of the results which we obtained through the different testing approaches. Consequently, we outline the abstract comparison methodology. Finally in Section 5, we propose the "best practices" to effectively exploit the different techniques used.

2 Unit testing

Ideally tests are implemented in the flow of the software development process, which means at increasing granularity. Only when we have some module/class implemented and their unit test cases ready, can we advance creating integration tests or new features. For example, it is possible to proceed in a cyclical manner by first writing unit tests for a few classes, then developing the necessary integration tests for those classes, then restarting again with new classes and their unit tests, which leads to further integration testing, and restarting the cycle again until the system is completed. Within this process unit test cases fill the foundation of the whole testing system and other testing parts can rely on first checking.

The principal aims of unit testing are the same as functional testing: verify the behavior of the tested component and help in finding implementation defects. The difference consists in their target: unit testing is mainly focused on small source code parts which are separately testable. Unit test cases input small functions or methods with specific values and check the output against the expected results, verifying the correct code behavior.

2.1 Failure detection and regression avoidance

Unit testing is mainly used to achieve two objectives: searching for defects in new -or not previously tested- code and avoiding regression after the evolution of code already under test. In the first instance, through test cases, the test engineer tries to check if the code is correctly implemented and does what it is expected to do; in the second situation failing tests warn that a software functionality that was previously behaving correctly stopped working as intended.

For this reason, in literature and industry we find unit-test generation tools that address those two different issues: failure detection and regression avoidance. In the former case tools try to spot unwanted or unexpected behavior, which could lead to a detection of failure. In the latter case they generate "tests that capture, and help you preserve, the code's behavior" [4]. They will "pass", if the code behaves in the same way that it did at the moment of their creation, but will "fail" whenever a developer changes the code functionalities, highlighting unanticipated consequences.

As we focus essentially on tools which are based on random testing, the main difference between tools that create unit tests for failure detection purposes and tools that generate regression tests is how they deal with the *oracle problem*. "An oracle is any (human or mechanical) agent which decides whether a program behaved correctly in a given test, and accordingly produces a verdict of "pass" or "fail" [...] oracle automation can be very difficult and expensive" [3].

When automatic generation tools produce regression tests, the source code which is provided is supposed to be without defects. For this reason, those tools generally consider the system itself as a sort of oracle: everything it outputs in reply to an input is the expected and correct answer. Thus, they will create only not failing tests, and, in this way, they take a "snapshot" of the tested system state.

On the other hand, tools that generate unit tests that reveal bugs in code cannot consider the tested system as the oracle, because it could provide both right and wrong answers to inputs, as it is not supposed to be without defects. For this reason, these tools need another way to know if the received output is the expected one or not (i.e. to solve the oracle problem). The tools we examined in [7], use a different approach to tackle this issue. One of them [14] considers unexpected exceptions that are raised as evidence of a possible failure. It does not need the test engineer to provide any additional information before submitting the source code to

the tool. Another software [33] requires more information about the tested class, such as class *invariants*. These properties will be used as the oracle: if the tool discovers an instance in which the submitted code does not respect the requested invariants, a failing unit test will be created. If the test engineers want to use this tool, they have to spend time preparing a more detailed input, which actually constitutes the oracle.

Finally, it is important to underline that failure detection tests and regression tests have different targets and for this reason they are mainly created in a different way. However, from the moment unit tests are implemented, they become part of the same unit test suite. A test that is used now to reveal a defect, can be incorporated later in the test suite and used as a regression test. A regression test that is not failing now, could be an effective failure detection test in the future or in a different implementation.

This already gives a hint of "best practices" which explain how to exploit and integrate both tests which are automatically generated by tools with different targets and tests which are manually implemented.

3 The experimental study

In [7] we wanted to show which approach was the best one when the test engineer had to decide whether to use tools which automatically generate unit tests and how. We chose to apply both manual and automatic techniques to a real case study, to have the chance of showing authentic data.

3.1 The Freenet Project

The Freenet Project [13] peer-to-peer software (henceforth Freenet) was chosen as subject for experiment, because it had all the features which we were interested in and which we considered relevant for our experiment validity.

Freenet is free software which allows the publishing and obtaining of information over the Internet, avoiding censorship. To achieve this, the network it creates is completely decentralized and publishers and consumers of information are anonymous. Communications between network nodes are encrypted and are routed through other peers to make communication tracking extremely difficult.

The software was subject to three drastic changes which caused important modifications of Freenet functionalities and its development team. For this reason, Freenet assumed, over time, many of the characteristics that allows us to equate it with legacy software [36].

Like with legacy software, many of the developers who created Freenet are not working on the project anymore, even though much of the source code they implemented is still in use and form an important part of the application. Then there is a crucial lack of documentation both for old source code and for new pieces of it; furthermore this is an issue for both high and low level documentation. For this reason, to fully understand Freenet functioning, or even only little parts of it, it is necessary either to read the corresponding source code or to interact with the developers community. As with many legacy software, the few active developers are constantly busy and they put most of their effort into developing new functionalities rather than revising the existing codebase.

Tests are so important that "the main thing that distinguishes legacy code from non-legacy code is tests, or rather a lack of comprehensive tests" [18]. Also in this sense, Freenet is a legacy software: when we started the experiment it had only 14 test cases for the whole source code, which included more than 800 classes.

The main difference between legacy software and Freenet lies in the fact that the latter is still under heavy development and evolution. This aspect, however, does not have any relevance to our study, so we have chosen Freenet to represent not only software without a pre-existing significant test suite, but also legacy software in general.

Positively, Freenet is open source code and was developed in Java, which is a language that is fully supported by the majority of automatic unit test generation tools. In this way, we had the possibility of comparing the most and latest relevant examples of industry and academic research in this field.

Furthermore, even though the software had many aspects related to legacy software, the developing structure was modern and efficient. It had effective mailing lists and an active IRC channel populated by the most important project developers. The code was maintained in a functional Subversion repository and we were granted full privileges to it. Finally the infrastructure to insert JUnit tests was already prepared and there was also support for continuous integration and test.

The changes we described earlier correspond to release 0.3, release 0.5 and release 0.7. During our experiment we used the last release which was the official one.

3.2 Development environment

The development environment for the experiment has been the GNU/Linux operating system provided with SUN Java 5 SE, Eclipse Europa IDE, Ant build tool version 1.7, JUnit version 3.8. During manual tests implementation hardware performance has not been an issue, whereas we got serious benefits in computational time using a dedicated workstation (dual Intel Xeon 2.8Ghz processor, with 2GB of Ram) when generating tests with the automatic tools and calculating the value of some of the metrics.

The developer who implemented unit tests manually was

a graduate student with a reasonable (about three years) industrial and academic experience in Java programming. He also had a basic knowledge of *x*Unit, previously developed using SUnit (the SmallTalk *x*Unit dialect) [2].

3.3 Procedure

In order to gain a better knowledge of the Freenet source code, environment and community, we began the experiment dedicating three full-time working months (which corresponded to almost 500 hours) to the manual implementation of unit tests. Then we started studying automatic test generation tools and we created tests using them. At the end we performed the comparison between each test suite that was generated using different techniques.

The decision to start with the manual implementation of tests made it easier to decide which and how many classes to put in our study domain. We decided to concentrate our efforts on support "leaf" classes (i.e. without outgoing references), because they offer two important advantages: they do not have significant references to other classes in the project -which means they are easier and faster to study and comprehend- and they are support classes, very similar to library classes, so they are often referenced through the whole code, and this allowed us to see the testing effects spread as broadly as possible across the system.

At the end of the creation of manual unit tests, we had two different "snapshots" of the Freenet system, that we used as a basis for further testing. First we had the source code which was present at the beginning (i.e. without unit tests), in which we found certain bugs through manual testing; and we had the source code after manual testing, which was modified to remove bugs and included the regression tests which we had manually implemented.

In order to successfully compare manual unit test implementation and automatic unit test generation tools, we used these two source code snapshots in a different manner. The first "snapshot" (i.e. without any manual modification) was useful for comparing failure detection capabilities. Using the source code as it was before the manual work, we were able to compare the bugs which had been found throughout manual implementation with the bug set that was detected by automatic tools. This comparison was possible only by working on the same source code basis, and allowed us to effectively spot similarities, advantages or faults of different the techniques.

On the other hand, we used the second "snapshot" to compare regression tests, manually and automatically generated. We already had manually implemented regression tests for this snapshot (which result from the manual testing), so it was more appropriate to use tools to generate regression tests over the same code, to see if their regression avoidance capability was less effective, similar or even better than the manual implementation.

It is still possible to checkout the source code we used as a basis for our tests, using the official Freenet Subversion repository.

3.4 Manual unit test implementation

To manually create unit tests, we first had to understand what was expected from the code we wanted to test. A reasonable approach to achieve this understanding is to read the documentation that was accompanying the code. Unfortunately, as already mentioned, very often it was not present or very poor and outdated. For this reason, we usually had to directly study the source code under test. It is not a good practice to understand what the code is supposed to do by reading it, because if the code contains defects or the developer had a wrong understanding of the requirement, studying only the code does not help in finding the problem. In addition, often we were unable to understand the functionalities supported by the analyzed code. So we had to search external documentation to clarify well-known problems that were addressed by the code (e.g., the DSA algorithm implementation), or -even worse- we had to contact the Freenet developers for further explanations. For this reason, the lack of documentation we encountered created important problems in the speed of manual testing, which we could not correctly estimate at the beginning of our work.

When we finally achieved full and correct knowledge of the classes that we wanted to test, we started writing test cases. Even though there is some research addressing the issue [41], there is not any widely accepted method of conducting the manual implementation of unit test cases. For this reason we decided to follow well-known and broadly used non-formal testing principles [24, 36].

First of all, when implementing tests, we tried to adopt a different point of view from the one which was used by the original code developers. We created a series of test cases that were intended to "break" rather than confirm the software under test. We also deeply analyzed all data structures that were created or used in each tested class. We created test cases to verify that data did not lose its integrity when using classes' methods or algorithms.

We also performed boundary-value analysis, because it has been statistically proven to be capable of detecting the highest number of defects [24]. We checked the code using the highest and lowest possible values, and we also tried to use values slightly outside the boundaries. Confirming the statistics, we found relevant bugs using this technique.

Our tests also performed control-flow path execution, which consists of trying to execute the code in every possible path to check the behavior of functions when dealing with inputs from different subsets. We usually created one test case for each possible code branch, to improve tests

readability and facilitate future source code debugging.

We also did exception handling checking to verify if source code correctly deals with wrong inputs and methods usage.

Finally we created mock objects [42] to replace the objects used by the methods under test. This offered a layer of isolation and allowed us to check the code when dealing with specific results from external objects which we created the mock object for.

The failures we gradually found during the manual creation of unit tests formed a basis of knowledge that was useful when trying to find bugs in new tested classes.

At the end of this manual activities, we had defined the Freenet classes that formed the domain of our experiment. As mentioned earlier, their number was below our and Freenet developers' expectations. We supposed that it was possible to test at least all the classes in the freenet.support package, however the documentation problems considerably slowed the manual testing process.

As mentioned before (Section 3.3), after the manual testing process we obtained two different "snapshots" of the source code. The former, which was the source code basis without any testing, is used to compare the bugs that were manually found with the ones found through automatic tools, in order to see if they are complementary or overlapping. The latter, which was the source code after the manual testing phase (that is after the removal of defects and the creation of the manual unit-test suite), is used to compare the regression avoidance quality of the different techniques.

3.5 Automatic generation tools selection

We based automatic unit test generation tools selection on two criteria. First of all, we wanted to analyze the results of automatic tools both when dealing with regression avoidance and failure detection, because we supposed that their effectiveness would be different. Then we chose tools that were simple for developers to learn or use, as they would be more easily adopted in an industrial context.

For these reasons, we focused our attention on JUnit Factory [5] which addresses regression tests creation, JCrasher [14] which deals with failure detection and Randoop [31] that can be used for both problems.

JCrasher and Randoop, and presumably also JUnit Factory, are based on random testing, which consists of providing well-formed but random data as arguments for methods or functions and checking whether the results obtained are correct. This random approach has a few advantages [14]: it requires low or no user interaction (except when an error is found), and it is cost-effective. However, one of the major flaw of random testing is that very low probability sub-domains are likely to be disregarded by it [25]. To deal with this issue, the chosen tools are designed to easily cover shallow boundary cases, "like integer functions failing on negative numbers or zero, data structure code failing on empty collections, etc." [14]. Finally, we decided not to use tools that rely on formal specification of the system under testing (e.g., model-based testing [34, 43]), because such systems are not common in practice [9], especially for legacy systems. In addition, in our real case study even an external model (based on requirements, documentation or other sources) was not available.

3.6 JUnit Factory

JUnit Factory was developed at AgitarLabs, the research division of Agitar Software, and it was freely usable upon registration.

JUnit Factory was different from the other tools we used because it was proprietary and was only offered as a service. This means that the user can neither see the software source code nor download and use the binaries on his computer.

To use the JUnit Factory service, we had to download a specific Eclipse [17] plugin (the web interface version was not suitable for a large project, as it was only useful for quickly seeing JUnit Factory results on very small independent classes). This plugin allowed us to choose the classes which we wanted to automatically generate unit tests for. When the selection was completed, the whole Eclipse project was uploaded to Agitar server and put in a queue. Some minutes after (the waiting time was influenced by tested class characteristics, such as lines of code and complexity) the resulting unit tests were ready and downloaded into our Eclipse project and became part of it. JUnit Factory creates unit-test cases with a clear variable and method naming, in order to increase their readability . However, those tests still remain terse, as depicted in Listing 1.

```
public void
testBytesToBitsThrowsNullPointerException1()
   throws Throwable {
 byte[] b = new byte[2];
 b[0] = (byte)75;
 try {
  HexUtil.bytesToBits(b, null, 100);
  fail(''Expected
   NullPointerException to be thrown'');
  } catch (NullPointerException ex) {
   assertNull(
    "ex.getMessage()",ex.getMessage());
   assertThrownBy(
    HexUtil.class, ex);
  }
}
```

Listing 1. JUnit Factory-generated test case

From that moment, JUnit Factory tests could be easily run using the Agitar's executor. Even though they were very similar to JUnit tests, they could not be executed using the standard JUnit framework, because they made use of Agitar proprietary libraries.

This usage description shows two major shortcomings of this tool: the user, through a non-secure communication channel, must upload the code to remote server, where it could be stored and used for unspecified purposes; and the service does not generate fully compatible JUnit tests, and for this reason proprietary libraries and a binary program must be included in the tested software in order to run them.

In our experiment we used an open source software, so we had no security issues, but when dealing with the automatic creation of unit tests for a proprietary software, these are problems that could mean that it is may impede the use of JUnit Factory.

Finally, due to its proprietary nature, it was not possible to study JUnit Factory internal functioning, but we had to rely on Agitar public articles [11]: which stated that it was based on the same research which formed the basis for Randoop (see Section 3.8).

3.7 JCrasher

JCrasher generates tests which target failures in the tested code. It produces type-correct inputs in a random fashion and attempts to detect bugs by "causing the program under test to crash, that is, to throw an undeclared runtime exception" [14]. It is almost completely automatic: no human supervision is required except for the inspection of the test cases that have caused an error.

JCrasher is based on C. Csallner and Y. Smaragdakis academic research and its source code is completely available under the MIT license. The binary executable and its results could be used freely by a test engineer without any restriction.

The usage of this tool is straightforward, even though it requires to be familiar with Apache Ant [6], the popular Java-based software build system. The user must prepare an Ant makefile with a specific target to invoke JCrasher [14] and a simple text file containing the list of classes for which it must create failure detection tests. Then the tool will create JUnit test cases, in a directory specified in a property of the Ant makefile. Even though the JCrasher target is failure detection only, it produces both passing and failing unit tests.

The test creation is a heavy task from a computational point of view, and this caused memory issues (i.e. java.OutOfMemory exception raising) leading us to move our development to the dedicated workstation we described previously. In addition, resulting JUnit test cases are extremely terse (e.g., Listing 2) and they are generated in the same directory as the tested source code.

```
public void test121() throws Throwable {
 try{
  java.lang.String s1 = "";
  java.lang.String s2 =
   "\"\n\\.‘’@#$%^&/({<[|\\n:.,;";
  HTMLNode h3 =
   new HTMLNode("", "", "");
  h3.addAttribute(s1, s2);
 } catch (Throwable throwable) {
  throwIf(throwable);}
}
```

Listing 2. JCrasher-generated test case

For these reasons we decided to produce tests for one class at a time and this also allowed us to separate tests in different meaningful subdirectories.

The number of test cases which were created varied from class to class, but it was usually extremely high. For example it outputs 100.000 test cases for a single class, many of which were almost identical. The execution time for such a vast quantity of tests was not acceptable even for a very fast system (it took about twelve minutes to check only one class). For this reason we reduced the number of tests before integrating them into our Eclipse project, deleting tests that used inputs from the same subset.

Finally we decided to keep the JCrasher tests which past in order to verify whether passing tests could be effectively used as regression tests.

3.8 Randoop

Randoop (*Rand*om *T*ester for *O*bject-*O*riented *P*rograms) is the practical result of research by Dr. M.D. Ernst and C. Pacheco and, like JCrasher, it is distributed under the MIT License. Randoop can generate unit tests to do both regression avoidance and failure detection.

In the first case the generation requires only a very low human interaction: in order to obtain the regression test suite, it is sufficient to specify which classes have to be tested, and their *helper classes*. Helper classes are needed because the tool will only generate tests using the specified classes. For example, in order to effectively test Collections, it is necessary to input a class (e.g., java.utils.TreeSet) that allows the tool to instantiate concrete collections. This class is defined as *helper class*.

Randoop then accepts a time limit which is an upper bound for the time it uses for its computations. We found that ten seconds was long enough to generate meaningful regression tests. We noted that longer time limits did not result in enhanced tests, but only lead to a higher number of tests, which are more expensive when carried out. Finally,

as Randoop is based on random input generation, its developer also offers the possibility of feeding Randoop with different random seed in order to obtain different random inputs. We note, however, that this possibility did not significantly change the produced tests, even though we tried a lot of different seeds. It is an issue that was also later noted by Randoop creator in [32], when he was trying to use it in another real world case. Randoop produces succinct test cases, where could even be difficult to spot the tested class (e.g., Listing 3).

```
public void testclasses15()
 throws Throwable {
  boolean v0 = false;
  freenet.support.SimpleFieldSet v1 =
    new SimpleFieldSet((boolean)v0);
  String v2 = "";
  double v3 = (double)1.0;
  double v4 =
   v1.getDouble((String)v2,(double)v3);
  String v5 = "hi!";
  long v6 = (long)100L;
  v1.put((String)v5,(long)v6);
  assertEquals(
   (double)1.0,
   (double)(Double)v4);
}
```

Listing 3. Randoop-generated regression test

On the other hand, Randoop failure detection functionality requires more time to prepare its input, because the user must write *contracts*. A contract is the way in which the programmer defines which are the properties that should remain the same in the class under examination. In practice, a contract is the implementation of a Java interface in which the developer makes some assertions about the tested object state. These assertions have to be always true. If during class checking, Randoop finds a sequence of method calls and inputs that make a check fail, then the test case which caused the failure is displayed.

We chose Randoop because we believe that a tool is more useful for developers, if they can express themselves using constructs they already know. In fact with this tool, they can write contracts in Java language. Other solutions [29], on the contrary, require the learning of a new formal language to express tested class properties and invariants. Randoop authors admit that this approach could be less expressive than using a specific formalism [31], but in our experience this did not turn out as a limiting factor.

Randoop performances were increased using the dedicated workstation, but its impact was less evident than when creating tests using JCrasher. Randoop did not suffer the same memory problem as JCrasher and it was reasonable to use it on the computer which we also used for standard programming. This implies that a single developer could benefit from using it, without the need of a powerful hardware.

As with JCrasher, we keep the tests that found failures to use them as regression tests. However, there were only a few of them and their influence was irrelevant.

3.9 Validity

The procedure we used to conduct the experiment led to relevant results [7], however since we are trying to outline an abstract method for unit tests comparison, we would like to suggest some enhancement that could be adopted to increase both validity of results and efficacy of comparisons.

In our experiment the same individual implemented manual unit tests and produced tests throughout automatic tools usage. To reduce the possibility of any influence between these two experiment phases, we first conducted the manual implementation then the automatic tools usage. In this way, we avoided bugs that were found by automatic test generation tools suggesting which tests the engineer could manually implement. What is more, the fact that we first conducted the manual part did not influence the automatic generation (and the experiment results confirmed this), because the human interaction was extremely low in the automatic phase. The only exception was when writing Randoop contracts. However, in this case the programmer did not have to directly write unit tests or give suggestions to the tool about how to write them. He only had to express class properties from a higher level of abstraction. For this reason, the fact that the manual part was done first, influenced only the time necessary to write contracts. The developer already knew the intended behavior of classes, so it was sufficient to express this in a more formal way.

In addition, by assigning the same individual to do both the tasks, we were able to see to what extent his normal abilities could be overcome or helped by the usage of automatic test generation tools. If we had assigned two different people, one for the manual task and one for the automatic tools usage, we would not have been able to be sure whether their personal programming abilities had influenced their results.

For this reason, the only way we could suggest improving the procedure of comparing different testing technique results, is to use a double-blind trial where professional developers are asked to implement manual and automatic tests to a set of representative software components. In this case a high number of developers is suggested, in order to lower differences caused by their respective testing abilities. It would be even better to ask each developer to test different parts of the software, using a different technique for each piece of code. In this way it would be possible to see the results of different testing techniques (without any in-

fluence from previous testing of the same code with other techniques) and not to be concerned about different developers abilities.

4 Comparison

After we obtained the unit tests through manual implementation and automatic tools usage, we proceeded to compare the effectiveness of each technique. Here we present the criteria we used to conduct the comparison and the results. In this way we attempt to show how these criteria can be enhanced and extended in order to outline an abstract methodology to perform a practical unit test effectiveness comparison.

We needed universal metrics capable of showing differences in a deterministic manner, in order to perform a correct empirical comparison and link each testing technique with its costs. In addition, such metrics are not only useful for comparisons, but also to help the test engineer to define the quality standard that the tests must achieve.

The quality of tests depends on many facets, for this reason we present the different metrics we used to capture as many characteristics as possible.

4.1 Time metrics

The first metric that we used to compare unit test creation techniques was **generation time**. As testing can consume more than fifty percent of software development time [9], this metric is crucial, especially considering the fact that the first target of automatic test generation techniques is often to dramatically reduce tests production time.

In this generation time metric we combine the time which is necessary to perform various aspects of tests generation. When considering the generation time of manually implemented tests, we also included the hours that were needed to correctly understand the classes we wanted to test and the amount of time we spent on them in order to perform source code refactoring or to improve the existing documentation. We decided to consider the time which was required to understand classes under test only for the manual implementation, even though it was also useful for Randoop contracts implementation. In the latter case only a high level class knowledge was necessary, which was faster and easier to achieve than the deep knowledge required to implement manual tests.

The generation time for automatic tools also includes the hours of effort required to learn their correct and most effective usage. JUnit Factory learning time was really short: it had an efficient documentation and it was straightforward to use. JCrasher had some useful examples which illustrated its usage in a easy way, for this reason it is quite simple to understand how to generate tests through it. Finally, Randoop required the longest learning time, because its documentation was not complete and the user had to learn what contracts are and how to write them. Fortunately Randoop's author replied to our questions in a very short time and has since improved the documentation.

Automatic tools generation time also includes the amount of time we spent interacting with them, both before they generated tests, and afterwards when we were inspecting them. Randoop generation time includes the time, which we spent before creating tests, to find helper classes and to write contracts, and the time -after the test case generation- to verify if error revealing tests reported real failures. JCrasher generation time includes the little effort required to prepare tests and to check error-revealing tests after their generation. Finally JUnit Factory generation time does not include preparation or inspection time, because the Eclipse plugin did all the necessary preparation and the resulting tests were not failing.

Another metric which is time related is test **execution time**. Tests are run frequently, unit tests in particular should be executed every time the tested code is modified, in order to verify that new errors are not introduced. This is one of the core extreme programming practice [8] and is also used in continuos integration and testing techniques. The majority of unit tests should be fast enough to be executed very often on a common development computer. For this reason execution time is an important metric when dealing with unit tests.

Although this metric could be improved simply by using a faster hardware, if the number of test cases is exponential the resulting benefits are less important. In order to deal with this issue, various strategies -mainly based on running only a subset of the regression tests- are studied and suggested [37, 30].

In the case study, even though some test cases could take longer to execute, we noted that the average execution time of a single test case was almost the same. This also reflects the fact that each unit test case should only check a small portion of the tested class [24], and this is usually a fast task. For this reason, we decided not to report execution time, but the **number of unit test cases** created using the different techniques. This gives a more correct suggestion of actual test execution time in general.

Table 1 shows these two metrics (generation time and number of unit tests) by different approaches. We split the Randoop metrics in two in order to reflect its different usages. The number of unit tests generated with the regression avoidance feature is much higher than the number unit tests generated when performing failure detection. In addition, the latter phase required more work to prepare the test creation.

Approach	time (hours)	test case
Manual	490	160
JUnit Factory	15	1,076
Randoop (failure)	115	12
Randoop (regression)	50	5,928
JCrasher	35	10,000+

Table 1. Tests # and generation time, by approach

4.2 Regression tests metrics

One of the target of automatic test generation tools is the creation of reliable regression tests, capable of creating a sort of safety net which could warn of possible errors introduced while modifying the tested code.

The most popular metric used to measure regression tests "quality" is **code coverage**, which is also the easiest metric to use. The main idea is to measure how much of the source code is exercised by unit tests during their execution. This is a white-box metric because it examines the internal coverage of the source code, rather than considering it as a black box.

There are different code coverage types [10], and the main difference between them is the basic unit they use for coverage. Class coverage measures the number of classes that are visited by the test suite; method coverage is the percentage of methods executed, without considering the method size; block coverage considers code blocks as the basic unit; statement coverage (also known as line coverage) tracks the invocation of single code statements. Branch coverage, which can also be found as decision coverage, has a slightly different functioning because it performs its calculatations measuring which code branches are executed. That is, it shows whether the boolean value of a control structure is set to both true and false.

Even though these code coverage flavors show different and complementary information about the test suite.

```
public int foo(int a, int b) {
 if (a > b) {
  //many lines of code
  ...
  return 1; }
 else {
  //only a few lines of code
  return 2; }
}
```

Listing 4. Line code coverage

For example if we prepare a test that exercises only the first branch of Listing 4, the corresponding line coverage will be quite high, because of the relative length of that part.

freenet.support class	Code Coverage (%)			
	Manual	JU.F	Rand	JCrash
Base64	79.5	90.7	93.2	64.6
BitArray	71	97.3	87.7	39.8
HTMLDecoder	55.9	94.6	26.7	91.4
HTMLEncoder	71.1	100	85.8	100
HTMLNode	96.6	100	80.9	29.9
HexUtil	73.9	91.4	68.3	35.7
LRUHashtable	83	100	74.2	55.8
LRUQueue	83	100	88.9	74.2
MultiValueTable	84.8	100	73.9	10.5
SimpleFieldSet	53.8	99.8	64.6	10.1
SizeUtil	82.6	96.9	53.4	53.4
TimeUtil	94.8	100	55.2	45.9
URIPreEncoder	78.7	100	46.1	0.0
URLDecoder	67.5	100	46.5	62.4
URLEncoder	85.7	100	88.8	93.9
Average	*77.5*	*98.0*	*68.9*	*51.2*

Table 2. Line code coverage

However, in the same situation block and branch coverage will report a mediocre result. When they calculate it, they do not consider branch or block length.

At the beginning of this subsection we put the word quality in quotation marks, because there is much research [10, 40] explaining that code coverage cannot be considered as a serious metric to measure unit tests quality. Through code coverage we can only see which part of the source code is used during test execution, but not how it is actually exercised (i.e. meaningfully or not). In [7] we empirically proved that code coverage is not a correct quality metric for regression tests. In addition, the results of the analysis on "code coverage and defect density (defect per kilo-lines of code) show that using coverage measures alone as predictors of defect density (software quality/reliability) is not accurate" [40].

However, code coverage can be used as a "negative metric": through it we can correctly see which parts of the source code are not executed by tests. For this reason, it can be used to help selecting and prioritizing tests, especially when dealing with a big software, where the test engineer must be selective about what to test.

We showed that code coverage is not a reliable quality metric, thus we decided not to put much effort into finding and using exotic code coverage flavors. We decided to use the most simple one (i.e. line coverage) to actually see what we could expect, in terms of code coverage, from different testing techniques.

Table 2 shows the code coverage that was achieved by each technique. The highest coverage is obtained by JUnit Factory, with an average value of 98% and a standard devi-

ation of 3%. The other techniques results have a decreasing code coverage average value, but also an increasing standard deviation value. It is relevant to note that the average code coverage reached by JCrasher is more than 50%, even though the tool only produced failure detection tests.

The target of regression tests is to inform the software developer when a change in the tested code also produced unexpected side-effects. For this reason, the best way to verify regression test accuracy is to demonstrate whether they can spot a change in source code functioning. And this is what **mutation analysis** does. The idea of this metric is to introduce mutations in the tested code, for example changing operators (e.g., substitute "- -" with "++"), variables (e.g., reset them, invert boolean values) and in other parts that could be changed without affecting the code execution. A first practical implementation of this analysis was proposed in [12] and [26].

When a mutation is introduced and it changes the expected code behavior, we define the resulting source code as a mutant. If this behavior change is detected by a test case, it is said that the test killed the mutant.

A change in the code could produce a functioning that is not different from the original one, in this case the mutation creates an equivalent mutant.

```
public class Mutable() {
 public static int aMethod() {
  int counter, returnValue;
  counter = 0;
  while (true) {
   //some operations on returnValue
   counter++;
   if (counter==12)
    return returnValue; }
 }
}
```

Listing 5. A not mutated class

For example in Listing 5 we show the original class and in Listing 6 one of its equivalent mutant.

```
public class Mutable() {
 public static int aMethod() {
  int counter, returnValue;
  counter = 0;
  while (true) {
   //some operations on returnValue
   counter++;
   if (counter>=12)
    return returnValue; }
 }
}
```

Listing 6. An equivalent mutant

The effort needed to check if mutants are equivalent or not, can be very big even for small programs [19]. However, we only want to compare the results of different techniques applied on the same source code basis. For this, the same equivalent mutants will appear in the mutation analysis for each technique, influencing the results in the same manner for each technique, without introducing any bias towards our comparison validity. For this reason, we did not consider necessary to find equivalent mutants produced during the mutation analysis.

The mutation analysis process first considers the source code and the corresponding test suite that is not failing (i.e. it is a suite of regression tests). Later, it creates a single mutation inside the source code, increasing by one the total number of mutations created, then it runs the tests again to verify if the mutant is correctly spotted and killed. If it is, the number of killed mutants is increased by one. Then the process restores the starting source code and restarts the cycle again creating another mutation. It loops until the last possible mutation is applied and checked. At the end we obtain a percentage which is the the number of mutants killed divided by the total number of mutants produced. This value is known as mutation score. The mutation analysis is an extremely long and repetitive task, and it is not reasonable to conduct it manually. It is necessary to have an effective tool to automate the process. In our real case study, we used Jester [23] which can perform the mutation analysis on Java code and creates effective and readable reports. In these reports it not only summarizes the mutation score reached, but it also shows the mutants created and which of them were killed by the chosen test suite.

Using Jester it is also possible to choose which mutations to perform in order to obtain mutants. It is sufficient to setup a configuration file, in which the user can specify how a piece of code could be mutated.

```
...
%==%!=
%++%--
...
```

Listing 7. A part of Jester configuration

For example, in Listing 7 we read that each "==" check will replaced with a "!=" check, and each "++" operator will be replaced with a "- -" operator. Using this possibility, we can avoid turning Jester into "a very expensive way to apply branch testing" [27]. In fact, Offutt explained that "the power of mutation depends on the mutation operators that create mutants of the program [...] Experimental research has found that exchanging 0s and 1s turns out to be almost useless because any input will find them. This is known as an "unstable" operator. Replacing predicates gets branch testing, no more no less" [27]. In our experiment we tried a mix of different mutations to obtain the best results from this analysis.

freenet.support class	mutation score (%)			
	Manual	JU.F.	Rand.	JCrash.
Base64	73	58	49	0
BitArray	46	76	82	7
HTMLDecoder	37	37	14	6
HTMLEncoder	28	59	32	6
HTMLNode	94	98	70	10
HexUtil	59	73	57	0
LRUHashtable	91	91	87	0
LRUQueue	58	100	79	0
MultiValueTable	75	96	59	0
SimpleFieldSet	49	84	48	3
SizeUtil	57	97	24	0
TimeUtil	93	99	17	0
URIPreEncoder	19	82	19	0
URLDecoder	71	86	19	10
URLEncoder	67	87	54	7
Average	*61*	*82*	*47*	*3*

Table 3. Mutation Score

Table 3 shows the mutation score for each technique. JUnit Factory confirms the first position which it also reached in the code coverage comparison, with a significant score of 82. Other techniques follow in the same order as in the code coverage comparison. We want to stress that JCrasher tests have no value as regression tests: even though the code coverage that was reached was decent, those tests were not capable of correctly characterize the system. They did not notice any significant source code change and thus were not capable of killing the mutants introduced. In fact, JCrasher tests are generated to identify only failures, and cannot recognize wrong behavior in the semantic of the tested class. This result still confirms that code coverage must not be used as a quality metric for regression tests. Finally, when working with JCrasher, the test engineer could discard the large amount of unit tests that are not revealing faults, because they are not even useful as regression tests.

4.3 Failure detection tests metrics

In [7] we commented that it is really difficult to determine an objective method to measure the ability of tests to detect defects. The number of failures that were found is not a complete metric to asses test quality, because it is evident that different defects could have a different impact on the system. However, in [38] it is shown that it is impossible to generally classify the severity of a defect without knowing the context in which the application was used.

Usually "many programmers might say that a null dereference is worse than not using braces in an if statement" [38]. On the other hand, a logical error caused by a lack of braces could be more severe, and harder to track down, than a null reference.

Approach	Found bugs
Manual	14
Randoop	7
JCrasher	4

Table 4. Number of bugs found, by approach

For this reason, in [7], we considered the Freenet developers' opinion as a further metric. They evaluated the defects detected by the different techniques and they explained that they had almost the same relevance for the system integrity.

This opinion allowed us to show table 4, in which we directly compared the number of defects found to show how the different techniques handled this task. In the manual approach, we also considered the semantic bugs we found, even though they could not be found through automatic tests. They were errors in the semantic of the tested classes that could only be discovered by accurately reading and understanding the source code that was under test.

As a future work it would be interesting to make use of a static bug finding tool, such as FindBugs [20]. This tool, which uses syntactic bug pattern detection and a dataflow component, statically inspects the code to warn about possible defects. These warnings have different levels of priority, and this classification could be useful to give a suggestion about the importance of bugs that automatic unit test generation tools could find. The process should be this: we run FindBugs to classify the warnings in the source code, then we use tools to generate tests and we remove the bugs they reveal. Then we inspect the code again, using FindBugs, to see whether the tools were capable of spotting the same defects and what their relevance was. In this manner we obtain a sort of classification of the bugs that were found.

However, the most effective way of classifying defects relevance is still to study the context and rely on developers opinion. Fortunately, the number of bugs detected is usually not so high that they cannot be manually inspected and evaluated.

4.4 Side effects

The last aspect we considered in [7] when we performed the comparison, was the presence of testing side effects. During the manual test implementation, the test engineer was forced to completely understand the tested classes and methods. In our case, as the documentation was extremely poor, he had to deeply analyze the source code in order to create effective tests. This necessity was highly time consuming, but also led to important side effects.

During manual tests, the engineer noticed and fixed some class performance issues, he increased class readability by removing code duplication and by using better variable naming, and he even split one class into two classes in order to enhance the code reuse.

Moreover, the documentation was improved, not only because of the comments the engineer added, but also because he tried to keep his unit tests as clear as possible. In this way, each unit test was an efficient and updated example of the piece of code they tested.

Only Randoop forced the developer to obtain at least a higher knowledge of the tested code. Instead, the other tools could be used without knowing anything about the system under test. For this reason, all the important testing side effects were not present when using automatic test generation tools. This means losing an important part of the testing benefits and is an aspect that must be taken into account when comparing different testing methodologies.

4.5 A comparison methodology

After detailing how we did the experiment we conducted in [7], it is now possible to abstract a comparison methodology to effectively compare test suites generated through different techniques, and thus compare the techniques themselves.

The first point is to consider failure detection and regression avoidance as two different tasks that cannot be compared because they have completely different targets. A failure detection test could be later used as a regression test, and vice-versa, but in order to compare them we must use different approaches.

It is even possible to use two different systems to create failure detection tests. However, the approach we suggest is the same we used in our experiment. The idea is to use the same system to compare both failure detection and regression avoidance ability.

We suggest starting by using the techniques that generate tests to find defects, and to use them separately but always on the same source code basis. In this manner, we can see if there are overlapping detected faults and, at the same time, we are not influenced by other technique results. In addition, at the end of this phase, we definitively obtain a more correct source code, that we can use later to check tools which generate regression tests.

If the test engineer were able to interact with the developers of the tested code, they might find agreement on a common scale to classify the severity of failures. Then the test engineer should submit the defects he found with different techniques to the developers (in a blind trial manner). They must return the failures classified. This is the best way to classify the relevance of errors, otherwise if it is not possible to interact with developers, the test engineer could use a tool like FindBugs and proceed as we depicted in Section 4.3.

Approach	MS/#(Test cases)	Code Cov.	MS
Manual	0.38	77.5	61
JU.F.	0.08	98.0	82
Randoop	0.01	68.9	47

Table 5. Test case accuracy, by approach

After having classified the detected errors, it is straightforward to compare the different test suites: it is sufficient to use the number and relevance of failures.

Then the regression tests comparison could take place. It is reasonable to use the source code that was used for the preceding phase, but only after having removed all the errors revealed. Usually tools that create regression tests consider the code base as bug-free, and, for this reason, it is useful to try to remove them before with failure revealing techniques.

As for the preceding phase, the different techniques should be used separately but always on the same source code basis. The first metric to calculate is code coverage, which is useful to get a fast overview of created tests. It would also be interesting to use coverages other than line coverage, to see if some techniques are less capable of creating particular scenarios. Finally, mutation analysis has to be performed to obtain the correct mutation score for each test suite. This part is the most important for comparing regression tests, as it is based on a true quality metric.

After these two phases (failure detection and regression test quality comparison), we should continue considering the time metrics. For the generation time it is important to include every aspect that consumed time during test generations -from technique learning to code inspection-. Because there are techniques that greatly improves some of those parts, and this must be taken into account for a valid comparison. For execution time we still suggest not directly calculating it, for example checking the run time, since a faster hardware could dramatically change the values. On the contrary, we suggest considering the number of generated tests, which is a better metric to have a realistic idea of test execution time for every kind of computer. Usually unit tests check a little part of code and for this reason, on average, they take almost the same time to be executed.

Finally, side effects should be considered. Techniques that force the engineer to study the source code he wants to test take a longer time to be used, but they can produce enormous benefits other than the tests themselves. At the end of a comparison which is conducted in this way, the developer has enough information and data to consciously decide which technique is the best for his particular situation. He can also decide to use more than one technique. In the next section we will try to explain what we consider the

"best practices" in order to integrate and exploit different techniques with different targets.

5 Best practices

As testing is expensive and time consuming, it is highly desirable to specify successful procedures for doing so [9]. For this reason, after having shown how to compare different testing techniques in order to become aware of their strengths and weaknesses, we here outline a procedure to exploit them so as to improve the testing process.

It is generally agreed that the most effective approach is to combine different testing techniques [9, 15, 16, 39], because each technique could spot different types of faults, and could suffer from a *saturation effect* [22].

When dealing with a legacy system or a modern system which is not yet tested (like the one we used in [7]), we suggest starting with tests for leaf classes or functions (see 3.3). [18] interestingly advises starting from inflection points in order to spread testing effects both to classes that use the tested classes and those that are used by them. An inflection point is a narrow interface to a set of classes. Any change in a class behind an inflection point is either detectable at the inflection point, or inconsequential in the application. However, in [7] we confirmed that finding and understanding inflection points is a hard task when dealing with systems that are not well documented, because the test engineer has to learn the functioning of many parts of the system and might have to read a large amount of the source code. That is the reason why we suggest starting from "leaf classes", as they are simple to understand and the benefits from their testing are spread across each class that uses them.

When creating unit tests for a not yet tested class, it is reasonable to start from the detection of failure in order to remove all defects and only then create regression tests. The first technique to use should be the easiest one, which does not require the test engineer knowing class functioning internals. For example, considering the techniques we used in [7], JCrasher would be the best choice to start testing with. It requires no test preparation and it can easily find interesting bugs in the code. What is more, the only task that the test engineer needs to do is to check whether failing unit tests that JCrasher generated are correctly reporting real errors. By doing this the test engineer starts to gradually learn what the internal functioning and the meaning of the tested class are.

After this part, the developer should continue using techniques that only need a high level of knowledge of the class under test. For example, from our past experiment, we suggest Randoop, because in order to write contracts it is not necessary to know what the little details of the tested class are, but only to have understood the general meaning of it. In this way, more defects could be revealed and the test engineer further improves his knowledge of the class when he has to verify and remove them.

In [7] we did not use any automatic test generation technique which required a low level knowledge of the tested class, but they could be effective if used in this phase as they could help the engineer in writing test cases. Otherwise, he could directly move to manual implementation of tests. It is evident that this manual work is made much easier by the automatic phases. Here we suggest not only checking for implementation defects but also verifying code documentation and searching for semantic errors. These are testing side-effects that are not produced by automatic tool usage, but they are a fundamental result of testing.

All the unit tests generated in this first "detection of failures" phase could be kept to be used later as regression tests. In the case of automatic tools we suggest keeping only unit tests that revealed errors, because the comparison also proved that not failing tests were useful as regression tests. For example in [7] it was absolutely useless to maintain the thousands of not failing tests that JCrasher produced, because they were completely useless regression tests (as depicted by the very low mutation score they reached).

The second part should be dedicated to "regression avoidance". During the preceding phase we accurately checked the class we wanted to test, and thus we could be confident enough about its correctness. For this reason, we can move to generate regression tests for it. To integrate the tests we created during the preceding phase, there are two possibilities: integrating them only by manually writing regression tests, or by creating regression tests automatically using appropriate tools and then eventually completing them manually. At the time of writing, and taking into consideration the tools that are available, the choice should be based on a trade-off between test creation time and test execution time.

As depicted in the comparison in [7], the manual creation is much more time consuming, but has the advantage of requiring less unit test cases than automatic tools to reach a good mutation score. This implies a shorter time to execute the whole test suite.

On the other hand, automatic generation tools need a greater amount of unit tests than the manual implementation to reach a high score in the mutation analysis. This means that automatically generated tests will require a longer time to be executed, and this could create problems when using continuous testing and integration. In addition, the number of tests could be so high that it would be a problem to run them in a common development computer. Fortunately the research in this field is extremely active and tools are becoming more and more effective. For example Randoop, which is more modern than JCrasher, is able to automatically remove useless unit tests, and JUnit Factory -which is a commercial software with a bigger team working on it- is

capable of reaching a higher mutation score than Randoop with half the tests. Finally, it is always necessary to manually create unit tests to integrate specific scenarios that were not exercised by automatically generated tests.

When describing these "best practices", we assumed the existence of a program to be tested, but they can also be easily adapted for adoption in a *test driven development* (TDD) [21] process. TDD suggests writing automated unit tests before developing the corresponding functional code, in short and rapid iterations. For each small function of the production code, the developer must first implement a test which clearly identify and validates what the code should do. Then the code is only developed to make the test pass, without adding any additional functionality not exercised by the test. The last part of every TDD iteration is the refactoring of both the production code and the test code.

The automatic unit test generation tools, that are now available, can only be used with a preexisting production source code to test. For this reason, when using a TDD process, the first phase is to manually write unit tests defining the expected functionalities. It is not worth writing many unit tests, but just concentrate on the few cases that can exactly exercise the purpose of the production class that will be later implemented. Then, the class could be created following the TDD procedure correctly. Later, the first few tests that have already been created must be integrated using tools to generate failure detection tests, in order to check whether the class implementation is without defects. In this phase, as the TDD process requires a prior knowledge of the class to be tested, the developer could take advantage of any kind of failure detection tools, even though they require familiarity with the class they must create tests for. After all the detected bugs are removed, the user must create regression tests. This can be performed manually or using appropriate tools, and the decision must be based on the trade-off between execution time and creation time, which we introduced before in this section. This stage makes the final refactoring phase easier and faster to accomplish.

By using the procedures we outlined in this section, the practitioner will receive all the benefits that the different unit test generation techniques supply. It will be easier to take decisions about how to schedule time dedicated to the different techniques.

6 Conclusions and Future Work

This paper outlines a novel comparison methodology that can be used to analyze the effectiveness of different unit-test creation techniques. We first explained what unit testing is and that failure detection and regression avoidance are the two issues that it mainly addresses. Then, we showed a real case study in which we created different unit-test suites whose advantages, shortcomings and effectiveness we wanted to assess . In order to do this comparison, we used a methodology that was able to quantitatively give information about test quality. Consequently, we used this practical example to abstract a comparison methodology that could be used not only in this case, but with unit-test suites produced through any technique.

For example, in the real case we studied, we realized that the automatic unit-test generation tools which we chose are really fast, and that they can produce test cases for a large number of classes in a very short time, and that they scale much better than a manual implementation. We also proved that those tools can create trusthworthy regression tests, which reach a high code coverage and, more importantly, a significant mutation score. Moreover, they can help the test engineer find defects, by the creation of unexpected scenarios or by adding a further abstraction level to test creation.

At the same time, the comparison was capable of also spotting the serious disadvantages that these tools suffer when compared to the manual testing approach. First, they do not force the developer to study the code under test. This means not getting the benefits of an accurate analysis of the source code: which could result in finding semantic defects, performing source code refactoring and improving the documentation. In addition, manually created tests are much more readable and are clear examples of the code they test. Moreover, to characterize classes, automatic unit-test creation tools produce at least ten times more test cases than the manual implementation, and even more when finding defects. This could be a problem when employing continous integration and testing, especially if they are used in common development computers.

It emerged, from the real case study, that the comparison methodology was able to richly characterize all the techniques it analyzed. However, in this paper, we also proposed some additional improvements to this procedure, in order to obtain a further effectiveness enhancement and to make it possible to use it for any kind of system that needs a high-quality unit-test suite.

At the end, we also outlined an efficient procedure based on "best practices", that can be used by test engineers to exploit the benefits of different unit-testing techniques. Using the results from the comparison, they can determine every tools advantage, and they can thus follow the "best practices" which explains how inserting each technique in the testing process can obtain a positive integration and relevant improvements.

A future work can involve the creation of a tool which automatizes the measurement of regression test quality (based on code coverage and mutation score) and helps integrate regression tests from different suites. The tool should report not only the scores, but also all the mutants created, with the unit-test cases -from all the different techniques-

that are able to kill them. In this way, the test engineers can adopt the most effective test suite as a basis, and they can integrate it with specific test cases from other techniques, which spot mutants that were not killed by the chosen main suite. Consequently, the effectiveness of the main regression test suites receive a significant improvement, without adding redundant unit-test cases.

Another area of future work is to investigate to what extent the human factor hinders the full potential of automatic unit-test generation tools, especially when the user has to inspect the results of the tool (e.g., in order to verify the failing unit-test cases) or to provide some input to further assist it (e.g., writing the class *contract*).

References

[1] JUnit. http://junit.org/. Accessed May 2009.

[2] SUnit. http://sunit.sourceforge.net. Accessed May 2009.

[3] A. Abran and J. W. Moore. *Guide to the Software Engineering Body of Knowledge (SWEBOK)*. IEEE, USA, 2004.

[4] Agitar Technologies. AgitarOne JUnit Generator. http://www.agitar.com/pdf/ AgitarOneJUnitGenerator-Datasheet.pdf. Accessed May 2009.

[5] Agitar Technologies. JUnit Factory. http://www.agitar.com/news/pr/20071015.html. Accessed May 2009.

[6] Apache Software Foundation. Apache Ant. http://ant.apache.org/. Accessed May 2009.

[7] A. Bacchelli, P. Ciancarini, and D. Rossi. On the effectiveness of manual and automatic unit test generation. In *ICSEA '08: Proc. of The Third Int'l Conf. on Software Engineering Advances*, pages 252–257. IEEE Computer Society, 2008.

[8] K. Beck. Embracing change with extreme programming. *Computer*, 32(10):70–77, 1999.

[9] A. Bertolino. Software testing research: Achievements, challenges, dreams. In *FOSE '07: 2007 Future of Software Engineering*, pages 85–103. IEEE Computer Society, 2007.

[10] C. Beust and H. Suleiman. *Next Generation Java Testing*. Addison-Wesley Professional, October 2007.

[11] M. Boshernitsan, R. Doong, and A. Savoia. From Daikon to Agitator: lessons and challenges in building a commercial tool for developer testing. In *ISSTA '06: Proc. of the 2006 int'l Symposium on Software testing and analysis*, pages 169–180. ACM, 2006.

[12] T. A. Budd. *Mutation analysis of program test data*. PhD thesis, New Haven, CT, USA, 1980.

[13] I. Clarke, S. G. Miller, T. W. Hong, O. Sandberg, and B. Wiley. Protecting free expression online with Freenet. *IEEE Internet Computing*, 6(1):40–49, 2002.

[14] C. Csallner and Y. Smaragdakis. JCrasher: An automatic robustness tester for Java. *Software – Practice & Experience*, 34(11):1025–1050, 2004.

[15] C. Csallner and Y. Smaragdakis. Check 'n' crash: combining static checking and testing. In *ICSE '05: Proceedings of the 27th international conference on Software engineering*, pages 422–431. ACM, 2005.

[16] M. d'Amorim, C. Pacheco, T. Xie, D. Marinov, and M. D. Ernst. An empirical comparison of automated generation and classification techniques for object-oriented unit testing. In *ASE '06: Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering*, pages 59–68. IEEE Computer Society, 2006.

[17] Eclipse Foundation. What is Eclipse and the Eclipse Foundation? http://www.eclipse.org/org/#about, May 2009. Accessed May 2009.

[18] M. C. Feathers. *Working Effectively with Legacy Code*. Prentice Hall, USA, September 2004.

[19] P. G. Frankl, S. N. Weiss, and C. Hu. All-uses vs mutation testing: an experimental comparison of effectiveness. *J. Syst. Softw.*, 38(3):235–253, 1997.

[20] D. Hovemeyer and W. Pugh. Finding bugs is easy. *SIGPLAN Not.*, 39(12):92–106, 2004.

[21] D. Janzen and H. Saiedian. Test-driven development: Concepts, taxonomy, and future direction. *Computer*, 38(9):43–50, 2005.

[22] M. R. Lyu, editor. *Handbook of software reliability and system reliability*. McGraw-Hill, Inc., Hightstown, NJ, USA, 1996.

[23] I. Moore. Jester – a JUnit test tester. In *Proc. of the 2nd Int'l Conf. on Extreme Programming and Flexible Processes*, pages 84–87, 2001.

[24] G. J. Myers. *The Art of Software Testing*. Wiley & Sons, USA, June 2004.

[25] S. C. Ntafos. On comparisons of random, partition, and proportional partition testing. *IEEE Transactions on Software Engineering*, 27(10):949–960, 2001.

[26] A. J. Offutt. A practical system for mutation testing: Help for the common programmer. In *Proc. of the IEEE Int'l Test Conf. on TEST: The Next 25 Years*, pages 824–830. IEEE Computer Society, 1994.

[27] A. J. Offutt. Jester analysis. http://cs.gmu.edu/ offutt/jester-anal.html, April 2005. Accessed May 2009.

[28] A. Oram and G. Wilson. *Beautiful Code: Leading Programmers Explain How They Think*. O'Reilly Media, Inc., June 2007.

[29] C. Oriat. Jartege: A tool for random generation of unit tests for Java classes. In *Proc. of 2nd International Workshop of Software Quality - SOQUA'05*, pages 242–256, September 2005.

[30] A. Orso, N. Shi, and M. J. Harrold. Scaling regression testing to large software systems. *SIGSOFT Software Engineering Notes*, 29(6):241–251, 2004.

[31] C. Pacheco and M. D. Ernst. Randoop: feedback-directed random testing for Java. In *OOPSLA 2007 Companion, Montreal, Canada*. ACM, October 2007.

[32] C. Pacheco, S. K. Lahiri, and T. Ball. Finding errors in .NET with feedback-directed random testing. In *ISSTA '08: Proc. of the Int'l symposium on Software testing and analysis*, pages 87–96. ACM, 2008.

[33] C. Pacheco, S. K. Lahiri, M. D. Ernst, and T. Ball. Feedback-directed random test generation. In *Proc. of the 29th Int'l Conf. on Software Engineering*, 2007.

[34] M. Pezzé and M. Young. *Software Testing and Analysis: Process, Principles, and Techniques*. John Wiley & Sons, 2008.

[35] S. L. Pfleeger. *Software Engineering: Theory and Practice*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001.

[36] R. S. Pressman. *Software Engineering: A Practitioner's Approach*. McGraw Hill Higher Education, USA, June 2001.

[37] G. Rothermel and M. J. Harrold. A safe, efficient regression test selection technique. *ACM Trans. Softw. Eng. Methodol.*, 6(2):173–210, 1997.

[38] N. Rutar, C. B. Almazan, and J. S. Foster. A comparison of bug finding tools for Java. In *ISSRE '04: Proc. of the 15th Int'l Symposium on Software Reliability Engineering*, pages 245–256. IEEE Computer Society, 2004.

[39] Y. Smaragdakis and C. Csallner. Combining static and dynamic reasoning for bug detection. In *Proc. Int'l Conf. on Tests And Proofs (TAP)*, volume 4454 of *LNCS*, pages 1–16. Springer, February 2007.

[40] K. Stobie. Too darned big to test. *Queue*, 3(1):30–37, 2005.

[41] D. Stotts, M. Lindsey, and A. Antley. An informal formal method for systematic JUnit test case generation. In *Proc. of the 2nd XP Universe and 1st Agile Universe Conf. on Extreme Programming and Agile Methods*, pages 131–143, 2002.

[42] D. Thomas and A. Hunte. Mock objects. *IEEE Software*, 19(3):22–24, 2002.

[43] M. Utting and B. Legeard. *Practical Model-Based Testing*. Morgan Kauffman, November 2006.

Incremental verification of consistency properties of large-scale workflows from the perspectives of control flow and evidence life cycles *

Osamu Takaki, Izumi Takeuti, Takahiro Seino, Noriaki Izumi and Koichi Takahashi
National Institute of Advanced Industrial Science and Technology (AIST)
2-41-6 Aomi, Koto-ku, Tokyo 135-0064, Japan
{o-takaki, takeuti.i, seino-takahiro, n.izumi, k.takahashi}@aist.go.jp

Abstract

We investigate consistency properties of workflows from the perspectives of control flow and evidence life cycles for incremental verification for large-scale workflows. For modeling complicated business processes in developing large-scale information systems, it needs to develop large-scale workflows that consist of a lot of small workflows. As a workflow becomes larger and larger, it becomes harder and harder to verify the workflow. Therefore, it is useful to verify large-scale workflows "incrementally", that is, to verify small workflows before they are integrated to form the large-scale workflows. However, in order to verify a workflow incrementally, it is necessary to consider consistency properties of not only a whole workflow but also a subgraph of the whole workflow. Thus, we extend the correctness property of acyclic workflows to that of acyclic workflows with multiple starts and/or ends. Correctness of workflows is one of the most important consistency properties for improving workflow quality from the control flow perspective. Extended correctness is a natural extension of the original correctness property and is preserved in the vertical composition and vertical division of workflows. We also define a consistency property for evidence life cycles in workflows with multiple starts. Moreover, in order to validate the consistency properties above for incremental verification, we investigate real workflows and explain how to verify the consistency properties by using an example.

Keywords: workflow, verification, correctness, evidence life cycle, incremental verification

1. Introduction

For developing large-scale information systems, it needs to model business processes that the systems support. A workflow, or a workflow diagram, is one of the most well known specifications for modeling of business processes. As business processes become more and more complicated, the workflows for modeling them become larger and larger. In the requirements analysis stage of developing large-scale information systems, for example, a number of engineers are needed for developing the workflows, which are divided into a number of smaller workflows. As a result, it has become harder for an engineer to verify the overall workflow in one operation. A method is thus needed for verifying large-scale workflows. One approach is to develop and verify workflows in parallel. We call such an approach "incremental verification". For incremental verification, small workflows should be verified before they are integrated to form a large scale workflow. However, in order to verify a workflow incrementally, it is necessary to consider consistency properties of not only a whole workflow but also a subgraph of the whole workflow, that can not completely satisfy the definition of a usual workflow. Thus, it needs to re-consider conventional consistency properties of a workflow from several perspectives.

Verifying the consistency of workflows from the control flow perspective is important, and several consistency properties have been defined and several verification methods have been developed. Correctness is one of the most standard consistency properties of acyclic workflows from the control flow perspective [10] (also [6] and [17]). However, these properties and methods can only be used to verify the overall workflow as a whole, not to incrementally verify workflows.

In this paper, we extend the correctness property to enable us to verify workflows incrementally. We consider workflows with multiple starts and/or multiple ends and extend the correctness of existing workflows to that of the extended workflows. A workflow in standard workflow lan-

*This work was supported by 'Service Research Center Infrastructure Development Program 2008' from METI and Grant-in-Aid for Scientific Research (C) 20500045.

guages, such as XPDL [23] and YAWL [19], has a single start and a single end. Verification of the consistency properties of workflow subgraphs requires consideration of workflows that may have multiple starts and/or multiple ends.

Extended correctness is a natural extension of the original correctness. Extended correctness is preserved in the vertical composition and division of the workflows. Extended correctness is a necessary and sufficient condition for obtaining a workflow with a single start and a single end. The workflow is correct in the sense of the original correctness property. It is obtained from a workflow satisfying extended correctness by appending appropriate workflows.

This paper is based on a previous one [13]. The main difference between them is the definition of a consistency property of evidence life cycles in a workflow with multiple starts and/or multiple ends. Here "evidence" means an annotation on a workflow, which denotes a document on which information is written, and/or with which something is approval, during the process of an operation. In [12] and [14], this property for a workflow with a single start is defined, based on "instances" of the workflow. We define the property for a workflow with multiple starts, by using "closed" subgraphs of the workflow. We define closed subgraphs of a workflow in this paper, while instances of a workflow are defined elsewhere [9]. Given this consistency property, one can incrementally check evidence life cycles in a workflow with multiple starts.

This paper also generalizes preservation theorems of vertical composition and workflow division (see Theorem 4.2 in this paper or ([13], Theorem 4.2)). This generalization, which is described in Theorem 4.4 in this paper, is easier to understand than that previously presented ([13], Appendix B).

The remainder of this paper is organized as follows. We define workflows with multiple starts and/or multiple ends and define vertical composition and workflow division in Section 2. We give a definition of an extended version of the original correctness property over acyclic workflows with a single start and a single end in Section 3. We refer to this correctness property as "extended correctness". We show the fundamental theorems of extended correctness in Section 4. We also give a definition of consistency of evidence life cycles in a workflow with multiple starts and/or multiple ends in Section 5. The definition is based on the previous one for a workflow with a single start [12]. We discuss the validity of extended correctness, consistency of evidence life cycles and incremental workflow verification based on the consistency properties in Section 6. Using an example, we investigate real workflows and explain how to incrementally verify control flow consistency for a large workflow. We discuss related work in Section 7 and summarize the key points in 8.

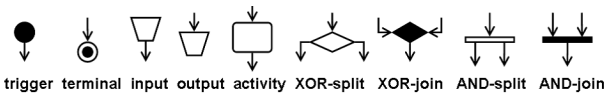


Figure 1. Shapes of nodes in workflows

2. Workflows

In this section, we define workflows. Moreover, we define certain composition and division of workflows. Workflows in this paper are essentially the same as those in previous studies such as [10], [6] and [17], except the point that a workflow in this paper may have multiple starts and ends. There are several languages of workflows with multiple starts and ends (see Section 7).

In this paper, we discuss workflows only on the control flow perspective. Therefore, we omit notions that are not relevant to control flow of workflows. For example, in this paper we do not consider data flow or actors in workflows.

Definition 2.1 (Workflows) A workflow denotes a directed graph $W := (\mathbf{node}, \mathbf{arc})$ that satisfies the following properties.

1. $\mathbf{node}$ is a non-empty finite set, whose element is called a node in W.

2. $\mathbf{arc}$ is a non-empty finite set, whose element is called an arc in W. Each arc f is assigned to a node called a source of f and another node called a target of f.

3. Each node is distinguished, as follows: trigger, terminal, input, output, activity, XOR-split, XOR-join, AND-split and AND-join.

   We employ the symbols in Figure 1 to describe nodes in a workflow in this paper.

4. Whenever an arc f has a node x as the target (or the source) of f, x has f as an incoming-arc (resp. an outgoing-arc) of x. The numbers of incoming-arcs and outgoing-arcs of a node are determined by the type of the node. We itemize them in the following table.

	incoming-arcs	outgoing-arcs
trigger, input	0	1
terminal, output	1	0
activity	1	1
XOR-, AND-split	1	$\geqq 2$
XOR-, AND-join	$\geqq 2$	1

Table 1. Numbers of incoming- and outgoing-arcs of a node

5. W has at least one start and at least one end.

6. For a node x in W, there exists a trigger or an input s and a path on W from s to x, where a path π from s to x denotes a sequence $\pi = (f_0, \ldots, f_n)$ of arcs in W such that the source of f_0 is s, the target of f_n is x and that the target of f_i is the source of f_{i+1} for each $i < n$. Moreover, there exists a terminal or an output e and another path on W from x to e.

Remark 2.2 In the previous paper [13], a workflow W is restricted to be a *connected* graph. That is, [13] assumes that, for each nodes x and y in W, there exists a sequence $(x_0, \ldots, x_n)$ consisting of nodes of W such that $x_0 = x$, $x_n = y$ and that there exists an arc in W between x_i and x_{i+1} for each $i < n$. However, in this paper, we also consider a workflow which is not connected. The reason why we consider some unconnected graphs as workflows is only because we have to consider unconnected workflows in Theorem 4.4 in Section 4. Actually, one can regard a workflow as a connected graph when they do not consider the theorem above.

Remark 2.3 In what follows, triggers and inputs are called "start nodes" or "starts". Moreover, terminals and outputs are called "end nodes" or "ends".

$\mathbf{WF}(n, m)$ denotes the set of all workflows with n starts and m ends and $\mathbf{WF} := \bigcup_{n,m} \mathbf{WF}(n, m)$. For a subgraph V of a workflow W, $\mathbf{arc}(V)$ denotes the set of all arcs in V, $\mathbf{start}(V)$ the set of all starts in V and $\mathbf{end}(V)$ the set of all ends in V.

We next define vertical composition and division of workflows.

Definition 2.4 (Vertical composition of workflows) Let $W_1, W_2 \in \mathbf{WF}$, $E \subset \mathbf{end}(W_1)$ and $S \subset \mathbf{start}(W_2)$. Moreover, assume that there exists a bijection f from E to S. Then, $W_1 *_f W_2$ denotes the workflow obtained from W_1 and W_2 by executing the following procedures.

(1) Remove all ends of E and their incoming-arcs.

(2) Remove all starts in S and their outgoing-arcs.

(3) For the source x of the incoming-arc of each end e in E and the target y of the outgoing-arc of each start $f(e)$ in S, add the arc from x to y.

$W_1 *_f W_2$ is called the vertical composition of W_1 and W_2 by f, and the arcs made in (3) above are called connecting-arcs from W_1 to W_2 by f.

For simplicity, in the remainder of this paper, we omit "f" in $W_1 *_f W_2$ and identify each $e \in E$ with $f(e) \in S$.

Example 2.5 The workflow in Figure 2 is the vertical composition of workflows $W1$ and $W2$, where the bijection function is expressed by two dot-lines in Figure 2, which maps e_1 to s_1 and e_3 to s_2.

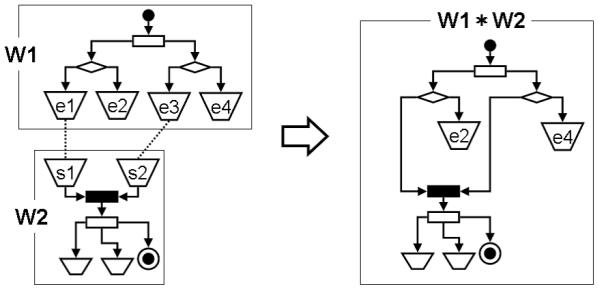


Figure 2. Vertical composition of workflows

Remark 2.6 In fact, all elements of E should be output nodes and those of S input nodes. However, it is not important to distinguish terminal nodes and output nodes (or trigger nodes and input nodes). Therefore, for simplicity, in the remainder of this paper, we assume that a start node denotes an input node only and an end node denotes an output node only, respectively.

Definition 2.7 (Vertical division of workflows) For a workflow W, if there exist workflows W_1 and W_2 with $W = W_1 * W_2$, then W is said to be vertically divided into W_1 and W_2.

3. Correctness and extended correctness

In this section, we explain correctness of workflows with a single start, which is defined in [10], and define an extended version of correctness, which we call extended correctness, and which is defined on workflows with multiple starts and multiple ends. Several basic theorems of extended correctness is shown in Appendix A.

In the remainder of this paper, we consider only acyclic workflows, which have no loop. In what follows, a workflow denotes an acyclic workflow.

Definition 3.1 For a workflow W and a start s in W, an instance of W from s denotes a subgraph V of W that satisfies the following properties.

(1) V contains s but does not contain any start except s. Moreover, for each $x \in V$, there is a path on V from s to x.

(2) If V contains an XOR-split c, then V contains a single outgoing-arc of c.

(3) If V contains a node c other than XOR-split, then V contains all outgoing-arcs of c.

For a workflow W, $\mathbf{INS}(W)$ denotes the set of all instances of W and $\mathbf{INS}(W, s)$ the set of all instances of W from s.

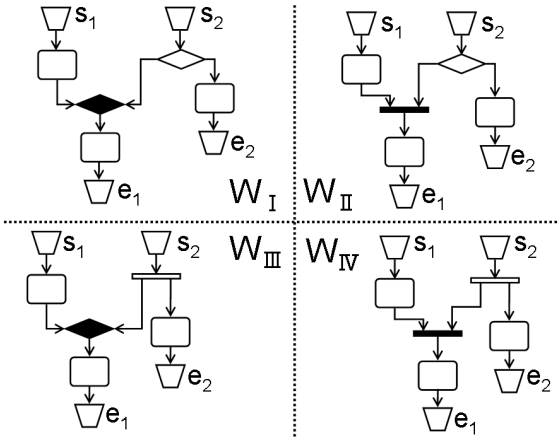


Figure 3. Four workflows

Example 3.2 We explain instances of workflows, by using the four workflows in Figure 3.

(1) The workflow $W_{\rm I}$ has three instances $U_1^{\rm I}$, $U_2^{\rm I}$ and $U_3^{\rm I}$, where $U_1^{\rm I}$ is the path from the start s_1 to the end e_1, $U_2^{\rm I}$ is the path from s_2 to e_1 and $U_3^{\rm I}$ is the path from s_2 to e_2.

(2) The workflow $W_{\rm II}$ has similar instances $U_1^{\rm II}$, $U_2^{\rm II}$ and $U_3^{\rm II}$ to those in $W_{\rm I}$.

(3) The workflow $W_{\rm III}$ has two instances $U_1^{\rm III}$ and $U_2^{\rm III}$, where $U_1^{\rm III}$ is the path from s_1 to e_1, and $U_2^{\rm III}$ consists of the path from s_2 to e_1 and that from s_2 to e_2.

(4) The workflow $W_{\rm IV}$ has similar instances $U_1^{\rm IV}$ and $U_2^{\rm IV}$ to those in $W_{\rm III}$.

Definition 3.3 Let W be a workflow.

(1) A subgraph V of W is said to be deadlock free if, for every AND-join r in V, V contains all incoming-arcs of r.

(2) A subgraph V of W is said to be lack of synchronization free if, for every XOR-join m in V, V contains a single incoming-arc of m.

Correctness is a consistency property of workflows in the viewpoint of control flow of them (cf. [10] or [17]). This property was defined on workflows with a single start and a single end in [10] and [17]. One can easily extend correctness into that over workflows with a single start and multiple ends by using instances of W. So, we consider correctness as a consistency property of a workflow that has a single start but may have multiple ends.

Definition 3.4 (Sadiq and Orlowska [10]) A workflow W with a single start is said to be correct if every instance V of W is deadlock free and lack of synchronization free.

From now, we extend the correctness property above for workflows with multiple starts and/or multiple ends. In order to define the extended correctness, we introduce some basic concepts.

Definition 3.5 For a workflow W and a non-empty subgraph V of W, V is said to be closed if each node x in V satisfies the following properties.

(1) If x is an XOR-split, then V contains a single outgoing-arc of x and the incoming-arc of x.

(2) If x is an XOR-join, then V contains a single incoming-arc of x and the outgoing-arc of x.

(3) Otherwise, V contains all incoming-arcs and all outgoing-arcs of x.

For a workflow W and a set of starts in W, $\mathbf{CL}(W)$ denotes the set of all closed subgraphs of W and $\mathbf{CL}(W, S)$ the set of all closed subgraphs V of W with $\mathbf{start}(V) = S$.

Note that, unlike instances of workflows, a closed subgraph of a workflow may not be connected as a graph.

Example 3.6 We explain closed subgraphs of workflows, by using the previous four workflows in Figure 3.

(1) All instances $U_1^{\rm I}$, $U_2^{\rm I}$ and $U_3^{\rm I}$ of $W_{\rm I}$ are also closed subgraphs of $W_{\rm I}$. Moreover, $U_1^{\rm I} \cup U_3^{\rm I}$ is an unconnected closed subgraph of $W_{\rm I}$.

(2) The workflow $W_{\rm II}$ has two closed subgraphs $U_1^{\rm II} \cup U_2^{\rm II}$ and $U_3^{\rm II}$ in Example 3.2.2.

(3) All instances $U_1^{\rm III}$ and $U_2^{\rm III}$ of $W_{\rm III}$ are also closed subgraphs of $W_{\rm III}$.

(4) $W_{\rm IV}$ has a single closed subgraph, that is $W_{\rm IV}$ itself.

Definition 3.7 Let W be a workflow.

(1) For $U_1, U_2 \in \mathbf{INS}(W)$, U_1 and U_2 are said to conflict on an XOR-split c if U_1 and U_2 share c but the outgoing-arc of c in U_1 differs from that in U_2.

(2) Let $\mathbf{U}$ be a set of some instances of W and c an XOR-split. Then, $\mathbf{U}$ is said to conflict on c if there exists a pair (U, U') on $\mathbf{U}$ that conflicts on c.

Definition 3.8 Let W be a workflow and S be a non-empty subset $\{s_1, \ldots, s_n\}$ of $\mathbf{start}(W)$. Then, S is called an inport of W if S satisfies the following properties: for each $U_i \in \mathbf{INS}(W, s_i)$ $(i = 1, \ldots, n)$, if $\{U_1, \ldots, U_n\}$ is not conflict on any XOR-split in W, then $U_1 \cup \cdots \cup U_n$ is closed.

Example 3.9 We explain in-ports of workflows, by using the previous four workflows in Figure 3.

(1) There are three non-empty subsets of $\mathbf{start}(W_{\rm I})$: $\{s_1\}$, $\{s_2\}$ and $\{s_1, s_2\}$ $(= \mathbf{start}(W_{\rm I}))$. $\mathbf{INS}(W_{\rm I}, s_1) = \{U_1^{\rm I}\}$ and $U_1^{\rm I}$ is a closed subgraph by Example 3.6.1. So, $\{s_1\}$ is an in-port of $W_{\rm I}$. Similarly, $\{s_2\}$ is also an in-port of $W_{\rm I}$. However, $\{s_1, s_2\}$ is not an in-port of $W_{\rm I}$, since $\{U_1^{\rm I}, U_2^{\rm I}\}$ is not conflict on any XOR-split in $W_{\rm I}$, but $U_1^{\rm I} \cup U_2^{\rm I}$ is not closed (cf. Examples 3.2.1 and 3.6.1).

(2) $W_{\rm II}$, too, has three non-empty subsets $\{s_1\}$, $\{s_2\}$ and $\{s_1, s_2\}$ of $\mathbf{start}(W_{\rm II})$. However, $\{s_1\}$ is not an in-port of $W_{\rm II}$, since $\{U_1^{\rm II}\}$ is not conflict on any XOR-split in $W_{\rm II}$, but $U_1^{\rm II}$ is not closed (cf. Examples 3.2.2 and 3.6.2). Similarly, $\{s_2\}$ is not an in-port of $W_{\rm II}$. Moreover, $\{s_1, s_2\}$ is not an in-port of $W_{\rm II}$, since $\{U_1^{\rm II}, U_3^{\rm II}\}$ is not conflict on any XOR-split in $W_{\rm II}$, but $U_1^{\rm II} \cup U_3^{\rm II}$ is not closed.

(3) $W_{\rm III}$ has two in-ports $\{s_1\}$ and $\{s_2\}$. However, $\{s_1, s_2\}$ is not an in-port of $W_{\rm III}$, since $\{U_1^{\rm III}, U_2^{\rm III}\}$ is not conflict on any XOR-split in $W_{\rm II}$, but $U_1^{\rm III} \cup U_2^{\rm III}$ is not closed (cf. Examples 3.2.3 and 3.6.3).

(4) $W_{\rm IV}$ has a single in-ports $\{s_1, s_2\}$ (cf. Examples 3.2.4 and 3.6.4).

Definition 3.10 Let W be a workflow and $\mathbb{I}$ a subset of the power set of $\mathbf{start}(W)$. Then, W is said to satisfy extended correctness for $\mathbb{I}$ if the following properties hold.

(1) $\mathbb{I}$ is a set of some in-ports of W.

(2) $\mathbf{start}(W)$ is covered with $\mathbb{I}$, that is, every $s \in \mathbf{start}(W)$ is contained in some element of $\mathbb{I}$.

We call the $\mathbb{I}$ above a covering in-port family of W.

Definition 3.11 A workflow W is said to satisfy extended correctness if W satisfies extended correctness for some covering in-port family.

Definition 3.12 Let W be a workflow.

(1) For an in-port I of W, the set $\{\mathbf{end}(V) | V \in \mathbf{CL}(W, I)\}$ is called the out-port family of W for I and denoted by $\mathbb{O}(W, I)$.

(2) For an in-port family $\mathbb{I}$ of W, the set $\{(I, \mathbb{O}(W, I)) | I \in \mathbb{I}\}$ is called the out-port assignment of W to $\mathbb{I}$ and denoted by $\mathbb{O}^*(W, \mathbb{I})$.

For the assignment $\mathbb{O}^*(W, \mathbb{I})$ of a workflow W to an in-port family $\mathbb{I}$, $\bigcup \mathbb{O}^*(W, \mathbb{I})$ denotes $\bigcup_{I \in \mathbb{I}} \mathbb{O}(W, I)$, that is the set of all out-ports of W for all in-ports in $\mathbb{I}$. That is, $\bigcup \mathbb{O}^*(W, \mathbb{I}) = \{\mathbf{end}(V) | V \in \mathbf{CL}(W, I)\ \&\ I \in \mathbb{I}\}$.

Example 3.13 We explain extended correctness of workflows and out-port assignments of them, by using the previous four workflows in Figure 3.
(1) By Example 3.9.1, $W_{\rm I}$ satisfies extended correctness for $\{\{s_1\}, \{s_2\}\}$. Moreover,

$$\mathbb{O}^*(W_{\rm I}, \{\{s_1\}, \{s_2\}\}) = \{(\{s_1\}, \{\{e_1\}\}), (\{s_2\}, \{\{e_1\}, \{e_2\}\})\}.$$

(2) By Example 3.9.2, $W_{\rm II}$ does not satisfy extended correctness.
(3) In the same way as $W_{\rm I}$, $W_{\rm III}$ satisfies extended correctness for $\{\{s_1\}, \{s_2\}\}$. Moreover,

$$\mathbb{O}^*(W_{\rm III}, \{\{s_1\}, \{s_2\}\}) = \{(\{s_1\}, \{\{e_1\}\}), (\{s_2\}, \{\{e_1, e_2\}\})\}.$$

(4) By Example 3.9.4, $W_{\rm IV}$ satisfies extended correctness for $\{\{s_1, s_2\}\}$. Moreover,

$$\mathbb{O}^*(W_{\rm IV}, \{\{s_1, s_2\}\}) = \{(\{s_1, s_2\}, \{\{e_1, e_2\}\})\}.$$

4. Fundamental theorems of extended correctness

In this section, we show fundamental theorems of extended correctness. These theorems are utilized for incremental verification for large-scale workflows. Proofs of the theorems in this section are shown in Appendix A.

The first theorem shows that extended correctness is a conservative extension of original correctness.

Theorem 4.1 For a workflow W with a single start, W is correct if and only if W satisfies extended correctness.

Theorem 4.1 insures that extended correctness adequate property to be a natural extension of original correctness.

We next show that extended correctness is preserved by vertical composition and division of workflows. For simplicity, we fix workflows W_1, W_2, a non-empty subset E_0 of $\mathbf{end}(W_1)$, a non-empty subset S_0 of $\mathbf{start}(W_2)$, and assume that there exists a bijection $f : E_0 \to S_0$. We also identify E_0 with S_0 and abbreviate the vertical composition $W_1 *_f W_2$ to $W_1 * W_2$.

We first show the theorem above in a special case (Theorem 4.2), and then show that in the general case (Theorem 4.4).

Theorem 4.2 Assume that $\mathbf{end}(W_1) = E_0 (= S_0) = \mathbf{start}(W_2)$ and let $\mathbb{I}$ be a covering family of $\mathbf{start}(W_1)$. Then, the vertical composition $W_1 * W_2$ satisfies extended correctness for $\mathbb{I}$ if and only if
(1) W_1 satisfies extended correctness for $\mathbb{I}$, and
(2) W_2 satisfies extended correctness for $\bigcup \mathbb{O}^*(W_1, \mathbb{I})$.

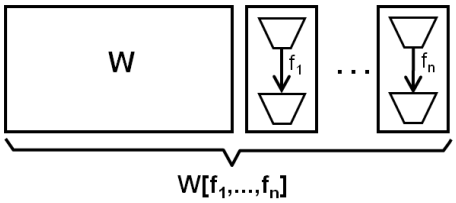


Figure 4. Obvious extension of a workflow

Definition 4.3 Let W be a workflow, and for $i = 1, \ldots, n$, let f_i be an arc with source a start node and with target an end node. Then, the *obvious extension* of W by $\{f_1, \ldots, f_n\}$, which is described by $W[f_1, \ldots, f_n]$, denotes the unconnected workflow obtained from W by adding arcs $f_1, \ldots, f_n$.

We illustrate $W[f_1, \ldots, f_n]$ by Figure 4.

Theorem 4.4 Let $S_2 := \mathbf{start}(W_2) - S_0$ [1] and $\mathbb{I}$ be a covering family of $\mathbf{start}(W_1) \cup S_2$. Then, the vertical composition $W_1 * W_2$ satisfies extended correctness for $\mathbb{I}$ if and only if
(1) W_1 satisfies extended correctness for

$$\{I \cap \mathbf{start}(W_1) | I \in \mathbb{I} \,\&\, I \cap \mathbf{start}(W_1) \neq \emptyset\}.$$

(2) W_2 satisfies extended correctness for

$$\{O \cap \mathbf{start}(W_2) \mid O \in \bigcup \mathbb{O}^*(W_1[f_1, \ldots, f_k], \mathbb{I}) \,\&\, O \cap \mathbf{start}(W_2) \neq \emptyset\},$$

where $f_1, \ldots, f_k$ denote arcs with source a start node and with target an end node (see Figure 5).

Theorems 4.2 and 4.4 claim that one can verify extended correctness of a workflow $W := W_1 * \cdots * W_n$ by calculating of the in-port families and the out-port assignments of $W_1, \ldots, W_n$. In the usual case, the calculation is not so complicated since most workflows have at most three start nodes (see Section 6).

Example 4.5 Consider W_{I}, W_{III} and W_{IV} in Figure 3. Then, by the function $f : \{e_1, e_2\} \to \{s_1, s_2\}$ with $f(e_1) = s_1$ and $f(e_2) = s_2$, one can consider nine vertical compositions $W_X * W_Y$, where X and Y are I, III or IV, respectively. By Example 3.13, $\bigcup \mathbb{O}^*(W_{\mathrm{I}}, \{\{s_1\}, \{s_2\}\}) = \{\{e_1\}, \{e_2\}\}$, $\bigcup \mathbb{O}^*(W_{\mathrm{III}}, \{\{s_1\}, \{s_2\}\}) = \{\{e_1\}, \{e_1, e_2\}\}$ and $\bigcup \mathbb{O}^*(W_{\mathrm{IV}}, \{\{s_1, s_2\}\}) = \{\{e_1, e_2\}\}$. Therefore, by Theorem 4.2, $W_{\mathrm{I}} * W_{\mathrm{I}}$, $W_{\mathrm{I}} * W_{\mathrm{III}}$ and $W_{\mathrm{IV}} * W_{\mathrm{IV}}$ satisfy extended correctness, but there is no other composition that satisfies extended correctness.

[1] For sets X and Y, "$X - Y$" denotes the difference set $\{x \in X | x \notin Y\}$.

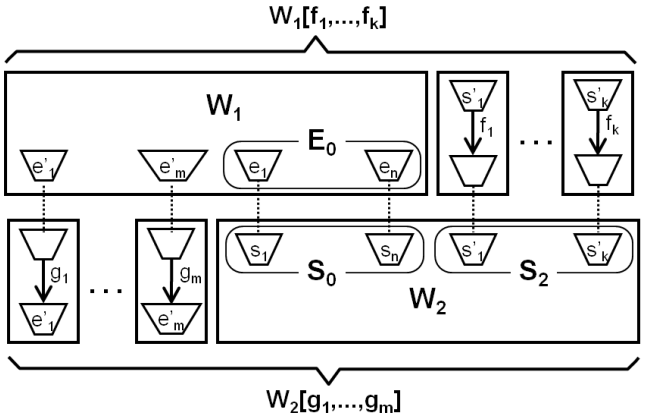


Figure 5. $W_1[f_1, \ldots, f_k] * W_2[g_1, \ldots, g_m]$ **(**$= W_1 * W_2$**)**

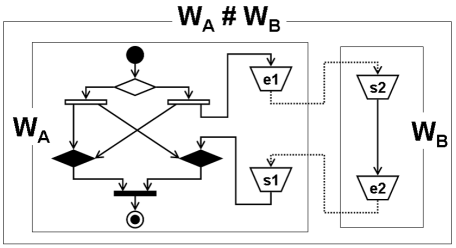


Figure 6. Another type of composition

On the other hand, since W_{II} does not satisfy extended correctness (see Example 3.13.2), one can obtain no workflow satisfying extended correctness by composing W_{II} and any workflow.

Remark 4.6 It is not a trivial problem whether a consistency property of workflows is preserved in certain composition or division of them. As an example, we give another composition $W_A \sharp W_B$ of workflows W_A and W_B in the way of Figure 6. Note that there exists control flow between W_A and W_B in both directions. While W_A does not satisfy extended correctness, $W_A \sharp W_B$ satisfies extended correctness. Therefore, the composition does not satisfy a property similar to Theorem 4.2 (or Theorem 4.4).

As the last part of this section, we define "extensible property" of workflows and show that the property is equivalent to extended correctness.

Definition 4.7 For a workflow W, W is said to be extensible if there exists a workflow W_0 such that $W_0 * W$ is correct.

Theorem 4.8 For a workflow W, W is extensible if and only if W satisfies extended correctness.

If a workflow W is extensible, it is possible that one can complete a correct workflow (with a single start) from W by extending W "vertically". On the other hand, Theorem 5.1 in [4] assures that one can modify a correct workflow W with a single start and multiple ends to that with a single start and a single end, which is essentially equivalent to W. So, Theorem 4.8 assures that, if W satisfies extended correctness, one can complete a correct workflow with a single start and a single end by extending W vertically and modifying the extended workflow in the way in the proof of Theorem 5.1 in [4].

Theorem 4.8 also assures that, if a workflow W does not satisfy extended correctness, one can not complete any correct workflow from W by extending W vertically. For example, since $W_{\rm II}$ in Figure 3 does not satisfy extended correctness, one can not complete any correct workflow from $W_{\rm II}$ by extending it vertically. This means that, if one likes to complete a correct workflow from $W_{\rm II}$, one has to modify $W_{\rm II}$ itself. So, it is useful to check extended correctness of an incomplete workflow (= a workflow with multiple starts and/or multiple ends) in the making of a correct workflow, since one may have an opportunity to modify structure of the incomplete workflow before it grows too large to modify the structure easily.

5. Consistency of evidence life cycles in a workflow with multiple starts

In the previous papers [12] and [14], we define a consistency property of life cycles of "evidences" in a workflow with a single start. We here define a similar consistency property for a workflow with multiple starts.

"evidence" is a technical term which means an annotation on a workflow, which denotes a document on which information is written, and/or with which something is approval, during the process of an operation. For simplicity, we often call such documents themselves "evidences". In large organizations such as large governments, evidences such as order forms, estimate sheets, invoices, and receipts play significant roles for purposes of feasibility, accountability, traceability, or transparency of business. Numerous actual operations are currently based on evidences even if they are carried out with information systems. Therefore, it is important to consider workflows in which one can concretely and precisely describe the life cycles of evidences to analyze requirements in developing large-scale information systems.

Roughly, the life cycles of evidences mean a series of states of the evidences, and consistency of evidence life cycles in a workflow means that the workflow has no inconsistent life cycles of evidences. In [12] and [14], we define a consistent property of evidence life cycles in a workflow with a single start, by using *instances* of the workflow. We here define that in a workflow with multiple starts, by using *closed* subgraphs of the workflow.

We precede the definition of the consistency property by that of evidences in a workflow.

5.1. Evidence

This subsection refers to [14]. We here regard an evidence as a paper document, which is composed, referred, re-written, judged, stored or dumped in some activities. Unlike data files, an evidence does not increase. Though one can make a copy of it, the copy is regarded not to be the same thing as the original evidence. Moreover, unlike data in a system multiple people can access simultaneously, an evidence can not be used by multiple people at the same time.

In the technical perspective, a list of evidences with length at least 0 is assigned to an activity, and an evidence E is defined to be a triple $(e, created, removed)$, where e is a label, and $created$ and $removed$ are boolean values. In what follows, we fix a non-empty set $\mathbb{E}$.

Definition 5.1 *Evidence* is a triple $(e, created, removed)$, where e is an element of $\mathbb{E}$ and $created$ and $removed$ are boolean values, that is, they are elements of $\{true, false\}$. For each evidence $E := (e, created, removed)$, we call e the *evidence label* of E.

Remark 5.2 For simplicity, we abbreviate evidences by the following ways.

(i) $(e, \mathrm{false}, \mathrm{false})$ is abbreviated to "e".

(ii) $(e, \mathrm{false}, \mathrm{true})$ is abbreviated to "$(-)e$".

(iii) $(e, \mathrm{true}, \mathrm{false})$ is abbreviated to "$(+)e$".

(iv) $(e, \mathrm{true}, \mathrm{true})$ is abbreviated to "$(+)(-)e$".

For a workflow W, we consider an allocation which assigns to each activity in W a string of evidences. Note that such an allocation may assign to some activities the empty string, i.e., the string with length 0. By using workflows, one can express a lot of workflows. In order to explain evidences, we give an example of a workflow which explains how to submit a paper, as follows.

For each workflow W, each activity A in W and for each evidence E in the string assigned to A, we call E an evidence *on* A and call A an activity *having* E.

Remark 5.3 In what follows, we assume that, for each workflow diagram W and each activity A in W, A does not have multiple evidences sharing the same evidence label. We call the condition *the basic evidence condition*.

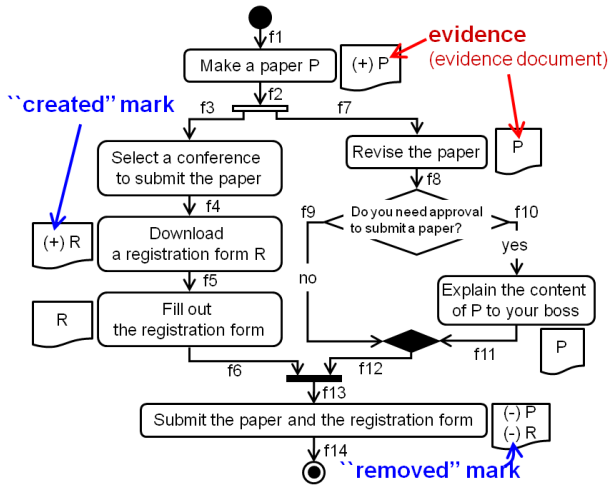


Figure 7. Workflow of paper submission

Since each workflow diagram W is assumed to satisfy the basic evidence condition, if an activity A in W has an evidence label e, A has just one evidence E with label e. So, we often say that e *is created (or removed) on* A if A has an evidence E having the $(+)$-mark (or the $(-)$-mark, respectively).

Example 5.4 In the workflow in Figure 7, a paper P is created on the activity "Make a paper P", and it is removed on the activity "Submit the paper and the registration form". The evidence also appears on the activities "Revise the paper" and "Explain the content of P to your boss".

5.2. Consistency property of evidence life cycles in a workflow with multiple starts

We here define a consistency of evidence life cycles in a workflow with multiple starts. This subsection also refers to [14].

Roughly, the "life cycle" of an evidence means that a series of states of the evidence. In order to define consistent life cycles of evidences in workflow in a rigorous manner, we introduce some new concepts.

Definition 5.5 For a workflow W, a *line* in W is a sequence of arcs in W

$$L = (A_1 \xrightarrow{f_1} A_2 \xrightarrow{f_2} \cdots \xrightarrow{f_{n-1}} A_n)$$

which satisfies the following properties.

(i) A_1 is an activity or the start in W.

(ii) A_n is an activity or an end in W.

(iii) $A_2, \ldots, A_{n-1}$ are nodes in W, each of that is not any activity, the start, nor any end.

For a line L above, A_1 is called the *source* of L, A_n the *target* of L and f_{n-1} the *target arc* of L.

Definition 5.6 A line L is said to be *equivalent* to another line L' if L and L' share the source and the target.

Example 5.7 The workflow in Figure 7 has 10 lines, as follows: $(f1)$, $(f2f3)$, $(f2f7)$, $(f4)$, $(f5)$, $(f6f13)$, $(f8f9f12f13)$, $(f8f10)$, $(f11f12f13)$ and $(f14)$.

Definition 5.8 A sequence π of lines is said to be *equivalent* to another sequence π' of lines if there exist lines $L_1, \ldots, L_n$ and $L'_1, \ldots, L'_n$ such that

$$\pi = (A_1 \xrightarrow{L_1} A_2 \xrightarrow{L_2} \cdots \xrightarrow{L_{n-1}} A_n)$$

$$\pi' = (A_1 \xrightarrow{L'_1} A_2 \xrightarrow{L'_2} \cdots \xrightarrow{L'_{n-1}} A_n)$$

and that, for each $i = 1, \ldots, n$, L_i is equivalent to L'_i.

$L \sim L'$ (or $\pi \sim \pi'$) denotes that L is equivalent to L' (π is equivalent to π', respectively). Note that every line is equivalent to itself, and so is every sequence of lines.

Definition 5.9 Let W be a workflow, V a closed subgraph of W and let e be an evidence in W. Then, the *consistent life cycle* of e on V is the sequence π of lines in V

$$\pi := (A_0 \xrightarrow{L_0} A_1 \xrightarrow{L_1} \cdots \xrightarrow{L_{n-1}} A_n)$$

which satisfies the following properties.

(i) Every activity A_i has e.

(ii) If A_0 is not the target of any line with source an input node, then e is created on A_0.

(iii) e is not created on A_i for any i with $0 < i \leqq n$.

(iv) If A_n is not the source of any line with target an output node, then e is removed on A_n.

(v) e is not removed on A_i for any i with $i < n$.

Definition 5.10 A workflow W is said to *have consistent evidence life cycles* if, for each closed subgraph V of W, each activity A in V and for each evidence e on A, there is an essentially unique consistent life cycle π of e on V which contains A.

The statement "there is an essentially unique consistent life cycle π of e on V containing A" means that there is a consistent life cycle π of e on V containing A and that $\pi \sim \pi'$ for each consistent life cycle π' of e containing A.

Example 5.11 The workflow in Figure 7 has two closed subgraph. The first closed subgraph (called U) consists of all nodes except the activity "Explain the content ..." and all arcs except $f10$ and $f11$. The second one (called V) consists of all nodes and all arcs except $f9$.

For an evidence P in Figure 7, U has just one consistent evidence life cycle of P: $((f2f7)(f8f9f12f13))$. V also has just one consistent evidence life cycle of P: $((f2f7)(f8f10)(f11f12f13))$. For another evidence R, U and V share the same consistent evidence life cycle of R: $((f5)(f6f13))$. Moreover, they have no other consistent evidence life cycle of R. Therefore, the workflow in Figure 7 has consistent evidence life cycles.

For the consistency property of evidence life cycles in a workflow with multiple starts, one can also have similar theorems to those in Section 4. Actually, one can easily show the following theorems.

Theorem 5.12 For a workflow with a single start, the original consistency property of evidence life cycles in the workflow (cf. [14], Definition 3.10) is equivalent to that in Definition 5.10.

Theorem 5.13 Let W_1 and W_2 be workflows, $E_0 := \{A_1, \ldots, A_n\} \subset \mathbf{end}(W_1)$, $S_0 := \{B_1, \ldots, B_n\} \subset \mathbf{start}(W_2)$, and f a bijection $E_0 \to S_0$ with $f(A_i) = B_i$. Then, $W_1 *_f W_2$ has consistent evidence life cycles if and only if so do W_1 and W_2 and, for any line $A \to A_i$ in W_1 and another line $B_i \to B$ in W_2, the following properties hold.

(i) For an evidence E on A, if E is not removed on A and if B is not any end node, then B also has E and E is not created on B.

(ii) For an evidence E on B, if E is not created on B and if A is not any start node, then A also has E and E is not removed on B.

The consistency of evidence life cycles in a workflow does not need extended correctness of the workflow. However, it is meaningless to define the consistency of evidence life cycles in a workflow which does not satisfy extended correctness. We show this claim by using a workflow W_{II}^* in Figure 8.

W_{II}^* has two closed subgraphs, both of which satisfies the conditions in Definition 5.10. So, W_{II}^* has consistent evidence life cycles. However, the structure of W_{II}^* is the same as that of W_{II} in Figure 3, and hence, W_{II}^* does not satisfy extended correctness. Actually, if the director returns the proposal P, the secretary can not receive it nor send it to the administration division. This example claims that, for a workflow which does not satisfy extended correctness, the semantics of the consistency of evidence life cycles in the workflow becomes ambiguous. Conversely, extended correctness of a workflow assures that the consistency of evidence life cycles in the workflow has the semantics we expect if one do not have to consider any set of start nodes which is not contained in the in-port family of the workflow.

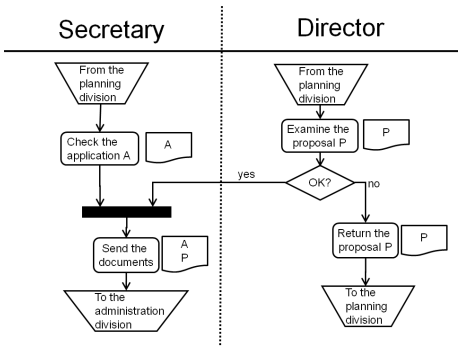


Figure 8. Wrong workflow

6. Discussion

In this section, in order to validate extended correctness, consistency of evidence life cycles and their fundamental theorems in Sections 4 and 5, we investigate real workflows and explain how to verify the consistency properties of a workflow by incremental verification.

6.1 Observations

We first investigate 154 workflows, which have been developed in requirement analysis for a real information system that helps one to manage personnel affairs. Each workflow has 10 to 30 nodes.

The observations are shown in the previous work [13].

Observation 1 Among the 154 workflows above, there are 101 workflows that have connections to other workflows. For example, there exists a large workflow that consists of 12 small workflows. [2] We describe the large workflow in Figure 9, where $W1, \ldots, W12$ describe the small workflows in the large workflow. In this figure, we simplify the small workflows. Especially, we omit all activity nodes in the small workflows.

We also classify 154 workflows on the numbers of their start nodes. Then, we have the following result. The result claims that, in most cases, the maximal in-port family and its out-port assignment of a workflow are not very large.

[2] We often consider a "large workflow" to be a set of workflows that have connections to one another.

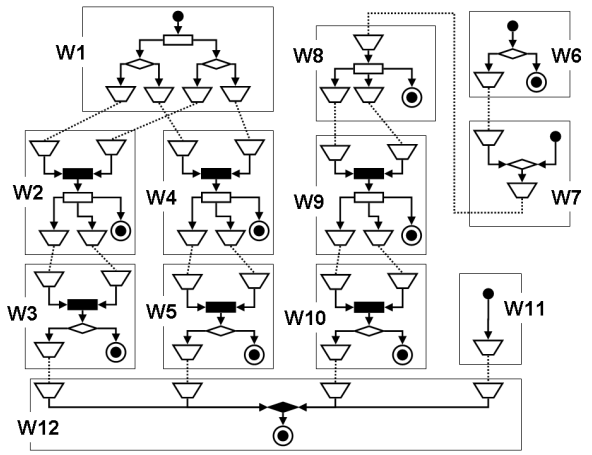


Figure 9. Large workflow W

number of start nodes	1	2	3	4	5	6	16
number of workflows	121	24	5	1	1	1	1

Table 2. Classification of 154 workflows on the numbers of start nodes

Observation 2 In most cases, even if two workflows are connected to each other, there is only one-way control flow between the two workflows. For example, while there is control flow from $W2$ to $W3$ in Figure 9, there is no control flow from $W3$ to $W2$. As far as the 154 workflows above, there are about 80 connections of two workflows, but, there are only 2 connections that have control flow between workflows in both directions (one can see an example of such a connection in Figure 6). Therefore, at least as far as the 154 workflows, vertical composition sufficiently covers connections between workflows.

Observation 3 As far as the 154 workflows, the order of development of the small workflows does not completely correspond to the direction of control flow of large workflows that consist of the small workflows. Moreover, there are some large workflows that plural engineers work together to develop. In fact, W in Figure 9 has been developed by two system engineers. In a case like this, incremental verification is especially useful, since it is possible that one of the engineers can obtain only an incomplete set of workflows in W.

Summary of the observations The first observation claims that about two thirds of the 154 workflows are connected to other workflows and that about one fifth of the 154 workflows have multiple starts. The second observation claims that, while there are about 80 pairs that are vertically composed, only two pairs are composed but not vertically composed. This means that 97.5 percent of all pairs that are composed are vertically composed. The last observation claims that the order of development of small workflows does not completely correspond to the direction of control flow of a large workflow consisting of them. By the observations, one can claim that it is meaningful to consider incremental verification for a large-scale workflow that consists of small workflows with multiple starts or vertically composed.

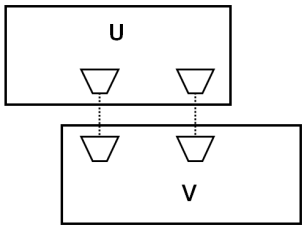


Figure 10. Workflows U and V

6.2 Application

In order to validate extended correctness, consistency of evidence life cycles of a workflow and the fundamental theorems of them, we here explain how to verify the consistency properties by incremental verification, by using two workflows U and V in Figure 10, that are vertically composed.

As we explained in the third observation in the previous section, U and V have been developed regardless of the control flow of $U * V$. For example, assume that only V has been developed. Dislike original correctness, one can verify control flow consistency of V based on extended correctness. Let V do not satisfy extended correctness. Then, by Theorems 4.2 and 4.4, whatever one develops U, $U * V$ will never satisfy extended correctness. This means that he/she should modify V at this point when V turn out not to satisfy extended correctness. Similarly, one can verify consistency of evidence life cycles of V even if V has multiple starts. He/She should also modify evidence life cycles of V at this point when V turn out not to have consistent evidence life cycles.

Moreover, assume that V has been modified to satisfy extended correctness (and to have consistent evidence life cycles) and that U has been developed additionally. If the workflow U does not satisfy extended correctness or consistency of evidence life cycles, then neither does $U * V$. Therefore, he/she should modify U at this point. If the workflow U satisfies extended correctness, he/she can know whether or not $U * V$ satisfies extended correctness, by checking conditions of the in-port families of V and the out-port assignments of U (see the comment immediately after Theorem 4.2). Similarly, If the workflow U has consistent evidence life cycles, he/she can know whether or not

$U * V$ has consistent evidence life cycles, by checking the conditions (i) and (ii) in Theorem 5.13.

As above, one can develop and verify the large workflow $U * V$ in parallel. Incremental verification is a useful approach, especially in the case of development of large-scale workflows.

7. Related work

The definition and fundamental theorems of extended correctness are based on the previous work [13]. In this paper, we show Theorem 4.4 by a simpler way than that of the similar theorem in [13].

There are a lot of researches of consistency properties of workflows in the viewpoint of control flow of them such as Aalst [15], Sadiq and Orlowska [9], Aalst [16], Sadiq and Orlowska [10], Verbeek et al. [21], Lin et al. [6], and Aalst et al. [17]. However, these researches deal with verification for a workflow with a single start and a single end as a complete workflow. In fact, a workflow in standard workflow languages such as XPDL [23] and YAWL [19] has a single start and a single end.

An open workflow net by Aalst et al. [18] can be considered to be a workflow with multiple starts and ends, and their "weak termination" essentially corresponds to soundness (correctness). Another well known workflow language EPC [3] has workflows with multiple starts and ends. By Mendling and Aalst [8], a semantics of EPC is given. Based on the notions above, one can obtain another extended correctness over (acyclic and cyclic) workflows with multiple starts and ends. However, the important point in this paper is that our extended version of correctness is a conservative extension of original correctness and it is preserved in vertical composition and division of workflows. These theorems are important for incremental verification based on the extended correctness. Since [18] and [8] have different purposes from ours, they do not show similar results about their extended correctness properties to our properties above. It is expected to show similar theorems based on correctness properties in [18] and [8].

By Dehnert and Aalst [2], Dongen et al. [20], and Mendling et al. [7], verification systems of consistency properties of workflows in EPC are developed. However, in order to verify consistency of workflows in EPC with the systems, users have to set start nodes which are fired at the initial point or the systems have to check all combinations of start nodes fired at the initial point. So, these approaches differ from ours.

A standard workflow model by Kiepuszewski et al. [4] also may have multiple starts and/or multiple ends. However, the semantics of the workflows in [4] is based on an assumption that a petri net modeling a workflow has a token on each initial place in every initial marking. Therefore, the correctness property of workflows defined in [4] is essentially the same as original correctness.

Kindler et al. [5] investigate "local soundness" for each sub-workflow in a workflow and "global soundness" for the whole workflow. The verification approach in [5] uses "scenario" that are used to verify global soundness of a workflow W from verification of local soundness of sub-workflows constituting W. So, the approach verifies a workflow based on necessary data for the workflow instead of the set of all sub-workflows of the workflow. Moreover, the ways to divide or compose workflows differ from ours.

Siegeris and Zimmermann [11] also investigate correctness properties of workflows to verify consistency of a whole workflow based on verifications of that of sub-workflows of the workflow. The verification approach for a workflow is based on verification for all sub-workflows constituting the workflow. Moreover, the ways to divide or compose workflows in [11] differ from ours, too.

On the other hand, in the previous papers [12] and [14], we investigate consistency of evidence life cycles in a workflow with a single start. We extend the previous work for a workflow with multiple starts.

Wang and Kumar [22] investigate document-driven workflow systems, where "documents " are essentially the same concept as evidences. They propose a framework for designing and managing workflows based on structures and states of documents. While our framework manages control-flow based workflows with evidences, their framework manages document-driven workflows. Thus, the meaning of the verification for their workflows differs from that of consistency of evidence life cycles in control-flow based workflows.

8. Conclusion

In this paper, we extend the results in our previous work [13], by adding consistency property of evidence life cycles in a workflow with multiple start nodes. The consistency property of evidence life cycles is based on that in a workflow with a single start node in [12].

The purpose of this paper is to develop an incremental verification methodology for large-scale workflows. As a basis for the verification methodology, we have defined extended correctness of an acyclic workflow with multiple starts and multiple ends. Extended correctness is a conservative extension of original correctness property over an acyclic workflow with a single start (Theorem 4.1). We also consider vertical composition and division of workflows, and show that extended correctness is preserved in these operations on workflows (Theorems 4.2 and 4.4). We also characterize extended correctness of a workflow as extensible property (Theorem 4.8).

In Section 5, we define a consistency property of evidence life cycles in a workflow with multiple starts, and show two fundamental theorems of the consistency (Theorems 5.12 and 5.13).

In Section 6, we investigate real 154 workflows in order to validate incremental verification for a large-scale workflow that consists of small workflows with multiple starts and/or vertically composed. Moreover, in order to validate extended correctness, consistency of evidence life cycles of a workflow and the fundamental theorems of them, we explain how to verify the consistency properties by using an example.

Since the workflow language in this paper is simple and conventional, one can apply the definitions and the theorems of the consistency properties in this paper for incremental verification for acyclic workflows in other languages such as BPMN [1] and XPDL.

Extended correctness, consistency of evidence life cycles and their theorems in this paper enable us to develop a concrete method to the consistency properties of large-scale workflows incrementally. Our next challenge is to develop a tool that helps one to verify large-scale workflows by the incremental methodology.

References

[1] Business Process Management Initiative (BPMI). *Business Process Modeling Notation (BPMN) Version 1.0*. Technical report, BPMI.org, 2004.

[2] J. Dehnert and W. M. P. van der Aalst. Bridging the gap between business models and workflow specifications. *International Journal of Cooperative Information Systems*, 13(3):289–332, 2004.

[3] G. Keller, M. Nuttgens, and A. W. Scheer. *Semantische Prozessmodellierung auf der Grundlage Ereignisgesteuerter Prozessketten (EPK)*. Technical Report 89, Institut fur Wirtschaftsinformatik Saarbrucken, Saarbrucken, Germany, 1992.

[4] B. Kiepuszewski, A. H. M. ter Hofstede, and W. M. P. van der Aalst. Fundamentals of control flow in workflows. *Acta Informatica*, 39(3):143–209, 2003.

[5] E. Kindler, A. Martens, and W. Reisig. Inter-operability of workflow applications: Local criteria for global soundness. In *Business Process Management: Models, Techniques, and Empirical Studies (BPM)*, LNCS 1806, pages 235–253. Springer, 2000.

[6] H. Lin, Z. Zhao, H. Li, and Z. Chen. A novel graph reduction algorithm to identify structural conflicts. In *Proceedings of the 35th Annual Hawaii International Conference on System Science (HICSS)*. IEEE Computer Society Press, 2002.

[7] J. Mendling, M. Moser, G. Neumann, H. M. W. Verbeek, B. F. van Dongen, and W. M. P. van der Aalst. Faulty epcs in the sap reference model. In *International Conference on Business Process Management (BPM)*, LNCS 4102, pages 451–457. Springer, 2006.

[8] J. Mendling and W. M. P. van der Aalst. Formalization and verification of epcs with or-joins based on state and context. In *Proceedings of the 19th International Conference on Advanced Information Systems Engineering (CAiSE)*, LNCS 4495, pages 493–453. Springer, 2007.

[9] W. Sadiq and M. E. Orlowska. On correctness issues in conceptual modeling of workflows. In *Proceedings of the 5th European Conference on Information Systems (ECIS)*, pages 943–964, 1997.

[10] W. Sadiq and M. E. Orlowska. Analyzing process models using graph reduction techniques. *Information Systems*, 25(2):117–134, 2000.

[11] J. Siegeris and A. Zimmermann. Workflow model compositions preserving relaxed soundness. In *Proceedings of 4th International Conference on Business Process Management (BPM)*, LNCS 4102, pages 177–192. Springer, 2006.

[12] O. Takaki, T. Seino, I. Takeuti, N. Izumi, and K. Takahashi. Verification algorithm of evidence life cycles in extended UML activity diagrams. In *Proceedings of The 2nd International Conference on Software Engineering Advances (ICSEA 2007)*. IEEE Computer Society Press, 2007.

[13] O. Takaki, T. Seino, I. Takeuti, N. Izumi, and K. Takahashi. Incremental verification of large scale workflows based on extended correctness. In *Proceedings of the 3rd International Conference on Software Engineering Advances (ICSEA 2008)*. IEEE Computer Society Press, 2008.

[14] O. Takaki, T. Seino, I. Takeuti, N. Izumi, and K. Takahashi. Verification of evidence life cycles in workflow diagrams with passback flows. *International Journal On Advances in Software*, 1(1), 2008 (to appear).

[15] W. M. P. van der Aalst. Verification of workflow nets. In *Application and Theory of Petri Nets 1997*, LNCS 1248, pages 407–426. Springer, 1997.

[16] W. M. P. van der Aalst. The application of petri nets to workflow management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.

[17] W. M. P. van der Aalst, A. Hirnschall, and H. M. W. Verbeek. An alternative way to analyze workflow graphs. In *Proceedings of the 14th International Conference on Advanced Information Systems Engineering (CAiSE)*, LNCS 2348, pages 535–552. Springer, 2002.

[18] W. M. P. van der Aalst, N. Lohmann, P. Massuthe, C. Stahl, and K. Wolf. From public views to private views: Correctness-by-design for services. In *Informal Proceedings the 4th International Workshop on Web Services and Formal Methods (WS-FM)*, LNCS 4937, pages 139–153. Springer, 2007.

[19] W. M. P. van der Aalst and A. H. M. ter Hofstede. YAWL: Yet another workflow language. *Information Systems*, 30(4):245–275, 2005.

[20] B. F. van Dongen, W. M. P. van der Aalst, and H. M. W. Verbeek. Verification of epcs: Using reduction rules and petri nets. In *Proceedings of the 17th Conference on Advanced Information Systems Engineering (CAiSE)*, LNCS 3520, pages 372–386. Springer, 2005.

[21] H. M. W. Verbeek, T. Basten, and W. M. P. van der Aalst. Diagnosing workflow processes using woflan. *The Computer Journal*, 44(4):246–279, 2001.

[22] J. Wang and A. Kumar. A framework for document-driven workflow systems. In *Proceedings of 3rd International Conference on Business Process Management (BPM)*, LNCS 3649, pages 285–301. Springer, 2005.

[23] Workflow Management Coalition (WfMC). *Workflow Management Coalition Workflow Standard: Workflow Process Definition Interface - XML Process Definition Language (XPDL)*. (WfMC-TC-1025), Technical report, Workflow Management Coalition, Lighthouse Point, Florida, USA, 2002.

A. Basic theorems and proofs of theorems

In this section, we show some basic theorems of extended correctness and theorems in Section 4.

Definition A.1 Let W be a workflow and C the set of all XOR-splits on W. Then, a phenomenon on W denotes a function $\psi : C \to \mathbf{arc}(W)$ satisfying that $\psi(c)$ is an outgoing-arc of c for each $c \in C$.

Lemma A.2 For a start s and a phenomenon ψ, there exists a unique instance for ψ from s, that is, there exists a unique instance V from s such that for every XOR-split c in V the outgoing-arc of c in V is $\psi(c)$.

Proof. Trivial. □

Lemma A.3 Let W be a workflow, s a start in W, π a path on W from s and ψ a phenomenon on W. Moreover, assume that, for an XOR-split c in W, if c is the source of an arc in π, then $\psi(c)$ is contained in π. Then, the instance of W for ψ from s contains π.

Proof. By induction on the length of π. □

Lemma A.4 For a workflow W, W is covered with $\mathbf{INS}(W)$, that is, every arc f in W is contained in some instance of W.

Proof. Let f be an arc in W. Then, there exists a path π with first element the outgoing-arc of some start s of W and last element f. So, by Lemma A.3, π is contained in some instance in $\mathbf{INS}(W, s)$. □

Lemma A.5 For a workflow W, every closed subgraph of W is deadlock free and lack of synchronization free.

Proof. Trivial. □

Proposition A.6 Let W be a workflow and $\{U_1, \ldots, U_n\}$ a set of some instances in W that is not conflict on any XOR-split in W. Then, $U := U_1 \cup \cdots U_2$ is closed if and only if U is deadlock free and lack of synchronization free.

Proof. By Lemma A.5, U is deadlock free and lack of synchronization free if U is closed. So, we assume that U is deadlock free and lack of synchronization free and show that U is closed.
(1) Let x be an XOR-split. Then, since x is contained in some instance U_i, the incoming-arc of x and some outgoing-arc(s) of x are contained in U. Moreover, since $\{U_1, \ldots, U_n\}$ is not conflict on any XOR-split in W, the outgoing-arc of x that is contained in U is single.
(2) Let x be an XOR-join. Then, since x is contained in some instance U_i, the outgoing-arc of x and some incoming-arc(s) of x are contained in U. Moreover, since U is lack of synchronization free, the incoming-arc of x in U is single.
(3) Let x be an AND-join. Then, since x is contained in some instance U_i and U is deadlock free, the outgoing-arc of x and all incoming-arcs of x are contained in U.
(4) Let x be another type node. Then, since x is contained in some instance U_i, all incoming-arcs and outgoing-arcs of x are contained in U. □

Lemma A.7 Let W be a workflow and I an in-port of W.
(1) For every $s \in I$ and $U \in \mathbf{INS}(W, s)$, there exists a closed subgraph V with $V \supseteq U$.
(2) For every $s \in I$ and $U \in \mathbf{INS}(W, s)$, U is lack of synchronization free.

Proof. (1) Let ψ be a phenomenon such that U is the instance for ψ from s. Then, for each $s_i \in I$ there exists the instance U_i for ψ from s_i. Since $\{U_1, \ldots, U_n\}$ does not conflict on any XOR-split, $U_1 \cup \cdots \cup U_n$ is closed. Moreover, for some $i \leqq n$, $s = s_i$ and hence $U = U_i$ by Lemma A.2.
(2) By (1) above, there is a closed subgraph V with $V \supset U$. Thus, we have the result since V is lack of synchronization free by Lemma A.5. □

Lemma A.8 For a workflow W satisfying extended correctness and a covering in-port family $\mathbb{I}$ of W, W is covered with $\bigcup_{I \in \mathbb{I}} \mathbf{CL}(W, I)$. Especially, W is covered with $\mathbf{CL}(W)$.

Proof. By Lemmas A.4 and A.7.(1). □

Lemma A.9 Let W be a workflow satisfying extended correctness and $\mathbb{I}$ a covering in-port family of W. Then, $\mathbf{end}(W)$ is covered by $\bigcup \mathbb{O}^*(W, \mathbb{I}) := \bigcup_{I \in \mathbb{I}} \mathbb{O}(W, I)$.

Proof. By Lemma A.8. □

Lemma A.10 For a workflow W with a single start, if W satisfies extended correctness, then W is correct.

Proof. Let W have only a single start s. Then, W has a single in-port $\{s\}$. So, by Def.3.8, every instance is a closed subgraph. Thus, by Lemma A.5, we have the result. □

Lemma A.11 Every correct workflow satisfies extended correctness.

Proof. Every instance of a correct workflow W is a closed subgraph of W. So, for the start s of W, $\{s\}$ is the in-port of W. □

Proof of Theorem 4.1 By Lemmas A.10 and A.11. □

Definition A.12 Let $W_1 \in \mathbf{WF}(n,m)$ and $W_2 \in \mathbf{WF}(m,l)$.
(1) Let V be a subgraph of $W_1 * W_2$. Then, $V\lceil W_1$ denotes the subgraph of W_1 uniquely obtained from $V \cap W_1$ by adding all possible ends in W_1 that correspond to connecting-arcs contained in V. Similarly, $V\lceil W_2$ denotes the subgraph of W_2 uniquely obtained from $V \cap W_2$ by adding all possible starts in W_2 that correspond to connecting-arcs contained in V.
(2) For a subgraph V_1 of W_1 and a subgraph V_2 of W_2, if there exists a subgraph V of $W_1 * W_2$ with $V\lceil W_1 = V_1$ and $V\lceil W_2 = V_2$, then V_1 and W_2 are said to be able to be composed, and $V_1 * V_2$ denotes the subgraph V above.

Lemma A.13 Let $W_1 \in \mathbf{WF}(n,m)$ and $W_2 \in \mathbf{WF}(m,l)$.
(1) For each $S \subset \mathbf{start}(W_1 * W_2)$ and each $V \in \mathbf{CL}(W_1 * W_2, S)$, $V\lceil W_1$ is a closed subgraph in $\mathbf{CL}(W_1, S)$ and $V\lceil W_2$ is that in $\mathbf{CL}(W_2, \mathbf{end}(V\lceil W_1))$.
(2) Conversely, for each $V_1 \in \mathbf{CL}(W_1)$ and for each $V_2 \in \mathbf{CL}(W_2, \mathbf{end}(V_1))$, if V_1 and $\mathbb{V}$ can be composed in $W_1 * W_2$, then $V_1 * V_2$ is a closed subgraph of $W_1 * W_2$.

Proof. Trivial. □

Lemma A.14 Let $W_1 \in \mathbf{WF}(n,m)$ and $W_2 \in \mathbf{WF}(m,l)$. If W_1 satisfies extended correctness for a covering in-port family $\mathbb{I}$ and if W_2 satisfies extended correctness for $\bigcup \mathbb{O}^*(W_1, \mathbb{I}) := \bigcup_{I \in \mathbb{I}} \mathbb{O}(W_1, I)$, then $W_1 * W_2$ satisfies extended correctness for $\mathbb{I}$.

Proof. We show that each $I \in \mathbb{I}$ is an image of $W_1 * W_2$. Let $I := \{s_1, \ldots, s_n\} \in \mathbb{I}$ and $U_i \in \mathbf{INS}(W_1 * W_2, s_i)$ $(i = 1, \ldots, n)$, and assume that $\{U_1 \ldots, U_n\}$ is not conflict on any XOR-split in $W_1 * W_2$. Then, $U_i\lceil W_1 \in \mathbf{INS}(W_1, s_i)$ for each $i \leqq n$, and $\{U_1\lceil W_1 \ldots, U_n\lceil W_1\}$ is not conflict on any XOR-split in W_1. Since I is an image of W_1, $U^1 := U_1\lceil W_1 \cup \cdots \cup U_n\lceil W_1$ is a closed subgraph of W_1.

On the other hand, $U^2 := U_1\lceil W_2 \cup \cdots \cup U_n\lceil W_2$ consists of instances in W_2, that have elements of $\mathbf{end}(U^1)$ as the starts. Moreover, the set of all the instances in W_2 above is not conflict on any XOR-split in W_2. Therefore, since $\mathbf{end}(U^1) \in \mathbb{O}(W_2, I) \subset \bigcup \mathbb{O}^*(W_2, \mathbb{I})$, U^2 is a closed subgraph of W_2. Since $U = U_1 * U_2$, by Lemma A.13.2, U is a closed subgraph of $W_1 * W_2$. So, we have the result. □

Lemma A.15 Let $W_1 \in \mathbf{WF}(n,m)$ and $W_2 \in \mathbf{WF}(m,l)$. If $W_1 * W_2$ satisfies extended correctness for a covering in-port family $\mathbb{I}$, then so is W_1 and W_2 satisfies extended correctness for $\bigcup \mathbb{O}^*(W_1, \mathbb{I})$.

Proof. We first show that W_1 satisfies extended correctness for $\mathbb{I}$. Let $I := \{s_1, \ldots, s_n\} \in \mathbb{I}$ and $U_i \in \mathbf{INS}(W_1, s_i)$ for $i \leqq n$. Moreover, assume that $\{U_1, \ldots, U_n\}$ is not conflict on any XOR-split in W_1. Then, there exists a phenomenon ψ_1 on W_1 such that each U is the instance for ψ_1 from s_i. So, we can have a phenomenon ψ on $W_1 * W_2$ such that the restriction of ψ to W_1 is ψ_1. Thus, for each $i \leqq n$, there exists the instance U_i^* of $W_1 * W_2$ for ψ from s_i. So, $U_i^*\lceil W_1 = U_i$ for each $i \leqq n$, and $\{U_1^*, \ldots, U_n^*\}$ is not conflict on any XOR-split on $W_1 * W_2$. So, since I is an image of $W_1 * W_2$, $U_1^* \cup \cdots \cup U_n^*$ is a closed subgraph of $W_1 * W_2$. Therefore, by Lemma A.13.1, $U_1 \cup \cdots \cup U_n$ is a closed subgraph of W_1. So, I is an image of W_1.

We next show that W_2 satisfies extended correctness for $\bigcup \mathbb{O}^*(W_1, \mathbb{I})$. Let E is an element $\{e_1, \ldots, e_m\}$ of $\bigcup \mathbb{O}^*(W_1, \mathbb{I})$ and $U_i \in \mathbf{INS}(W_2, e_i)$ for $i \leqq m$. Moreover, assume that $\mathbf{U} := \{U_1, \ldots, U_m\}$ is not conflict on any XOR-split in W_2. Then, there exists a closed subgraph V_1 in W_1 such that $\mathbf{start}(V_1) \in \mathbb{I}$ and that $\mathbf{end}(V_1) = E$. So, we can have instances $U_1^1, \ldots, U_k^1$ in W_1 such that $V_1 = U_1^1 \cup \cdots \cup U_k^1$. For each $j \leqq k$, we have the subset $\mathbf{U}_j$ of $\mathbf{U}$ by $\mathbf{U}_j := \{U \in \mathbf{U} : \mathbf{start}(U) \in \mathbf{end}(U_j^1)\}$. Then, for each $j \leqq k$, $U_j^* := U_j^1 * (\bigcup \mathbf{U}_j)$ is an instance of $W_1 * W_2$, and $\{U_1^*, \ldots, U_k^*\}$ is not conflict on any XOR-split on $W_1 * W_2$. So, since $W_1 * W_2$ satisfies extended correctness for $\mathbb{I}$, $U_1^* \cup \cdots \cup U_k^*$ is a summation in $W_1 * W_2$. Therefore, by Lemma A.13.1, $U_1 \cup \cdots \cup U_m = (U_1^* \cup \cdots \cup U_k^*)\lceil W_2$ is a summation in W_2, and hence, E is an image of W_2. □

Proof of Theorem 4.2 By Lemmas A.14 and A.15. □

Proof of Theorem 4.4 By the definition of a vertical composition (Definition 2.4), $W_1 * W_2$ is the same as $W_1[f_1, \ldots, f_k] * W_2[g_1, \ldots, g_m]$ in Figure 5 (see also Figure 2). So, the extended correctness of $W_1 * W_2$ for $\mathbb{I}$ is equivalent to that of $W_1[f_1, \ldots, f_k] * W_2[g_1, \ldots, g_m]$ for $\mathbb{I}$. So, by Theorem 4.2, it is equivalent to the following properties (i) and (ii).

(i) $W_1[f_1, \ldots, f_k]$ satisfies extended correctness for $\mathbb{I}$

(ii) $W_2[g_1, \ldots, g_m]$ satisfies extended correctness for $\bigcup \mathbb{O}^*(W_1[f_1, \ldots, f_k], \mathbb{I})$.

Now we first show that the property (i) above is equivalent to (1) in Theorem 4.4. For a subgraph V of $W_1[f_1, \ldots, f_k]$, V is closed in $W_1[f_1, \ldots, f_k]$ if and only if $V \cap W_1$ is closed in W_1. Thus, for an element I of $\mathbb{I}$ with

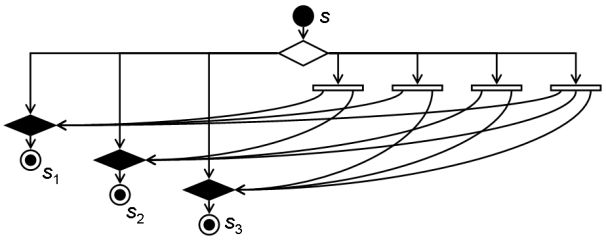


Figure 11. Workflow W_0

$I \cap \mathbf{start}(W_1) \neq \emptyset$, I is an in-port of $W_1[f_1, \ldots, f_k]$ if and only if $I \cap \mathbf{start}(W_1)$ is that of W_1. Therefore, since $\mathbb{I}$ is a covering family of $\mathbf{start}(W_1[f_1, \ldots, f_k])$, the property (i) is equivalent to (1) in Theorem 4.4.

The property (i) implies that $\bigcup \mathbb{O}^*(W_1[f_1, \ldots, f_k], \mathbb{I})$ is a covering family of $\mathbf{start}(W_2[g_1, \ldots, g_m])$. Therefore, when (i) holds, one can show that the property (ii) is equivalent to (2) in Theorem 4.2 in the similar way to the case of (i) above. Thus, the extended correctness of $W_1 * W_2$ for $\mathbb{I}$ is equivalent to the properties (1) and (2) in Theorem 4.4. □.

Lemma A.16 For a non-empty finite set S and a subset $\mathbb{S}$ of the power set of S with $S = \bigcup \mathbb{S}$, there exists a correct workflow W that has a single start s and that $\mathbb{S} = \mathbb{O}(W, \{s\})$.

Sketch of the Proof. Instead of showing this lemma directly, we give an example $S := \{s_1, s_2, s_3\}$ and $\mathbb{S} := \{\{s_1\}, \{s_2\}, \{s_3\}, \{s_1, s_2\}, \{s_1, s_3\}, \{s_2, s_3\}, S\}$ (=the power set of S), and illustrate a workflow W_0 satisfying the properties in this lemma for the S and $\mathbb{S}$ above by Figure 11. (All workflows satisfying the properties in this lemma can be constructed in similar forms to W_0.)

Each outgoing-arc of the XOR-split c in W_0 (see Figure 11) corresponds to an element of $\mathbb{S}$. Moreover, the numbers of outgoing-arcs of AND-splits $x_1 \sim x_4$ in W_0 correspond to the numbers of elements of $\{s_1, s_2\}$, $\{s_1, s_3\}$, $\{s_2, s_3\}$ and S, respectively. Note that outgoing-arcs corresponding to $\{s_1\}$, $\{s_2\}$ and $\{s_3\}$ do not have any AND-split, since $\{s_1\}$, $\{s_2\}$ and $\{s_3\}$ have a single element, respectively.

Obviously, W_0 is correct, and the ex-port family of W_0 for the in-port $\{s\}$ is $\mathbb{S}$. □

Proof of Theorem 4.8 By Theorem 4.2 and Lemma A.16. □

Crucial Service-Oriented Antipatterns

Jaroslav Král and Michal Žemlička
Charles University, Faculty of Mathematics and Physics
Department of Software Engineering
Malostranské nám. 25, 118 00 Praha 1, Czech Republic
kral@ksi.mff.cuni.cz, zemlicka@ksi.mff.cuni.cz

Abstract

Service-oriented architecture is now the most popular software engineering concept. Software related antipatterns – commonly used seemingly good but in fact wrong solutions – can have adverse consequences of varying importance. It implies that the use of an antipattern can and should be viewed as a risky event. It follows that some principles of risk management can be used. We propose a method based on slightly simplified procedures of risk management and assessment. Using the procedures we give a short list of the most risky antipatterns, i.e., antipatterns occurring very often and having crucial consequences and present principles of antipattern refactoring. We discuss the following crucial antipatterns: No Legacy (development from scratch), Standardization Paralysis, Business Process for Ever (Full Automation), Sand Pile (too fine grained services), On-Line Only (No Batch Systems). The discussion of antipatterns is based on a long-term experience with service-oriented and service-oriented like (e.g., process control) systems and on the analysis of practice. Contributions of the paper: evaluation of antipatterns as risky events, specification the properties of service-oriented systems in small firms and in e-government, differences between object-oriented and service-oriented antipatterns, requirements on service interfaces, and the list of the service-oriented antipatterns being the most important ones according to the evaluation.

Keywords: antipattern, risk management, SOA type, confederations, antipattern evaluation.

1. Introduction

We will discuss the antipatterns (wrong practices) in service-oriented architecture (SOA). We say that a system has service-oriented architecture if it is a virtual peer-to-peer network of loosely related software components having properties mirroring the behavior of real-world services. Technically such systems consist of components/services and middleware. The details and effects of such system vary. We will discuss this point in details.

Typical SOA systems are formed by a ”kernel” network of services providing main system capabilities. In enterprises they are often the services supporting manufacturing like inventory control, machine floor supervision, etc. The components can be for different SOA types of different sizes. They can be large legacy systems, large third-party products, or they can be almost all quite small software components (redeveloped) from scratch. The used interfaces and communication protocols can vary. The basic communication mode is asynchronous message exchange. Message formats can be programming oriented (based on remote procedure call technique and fine grained) or user domain oriented (e.g., XML-based messages mirroring user domain languages).

Service orientation and service-oriented architecture (SOA) are the leading edge of contemporary software engineering. Service orientation is a new paradigm for business-oriented software. In the area of real-time (process control) systems, the main principles of SOA are used for decades. Business is, however, different from technology, so business-oriented SOA systems have specific aims, users, and development practices. As such they require specific good practices and turns – patterns – that can be different from the ones known from object-oriented philosophy.

The paper is organized as follows: Basic facts on antipatterns, antipatterns as risks and risk management procedures, SOA architecture types called confederations and unions, list of most risky antipatterns in unions and confederations and their solution, differences between SOA antipatterns and object-oriented antipatterns.

2. Backgrounds

The practitioners collected an impressive collection of ”SOA wrong practices” and wrong solutions called antipat-

terns (compare [5, 4]).

A pattern [8] is a solution of some problem/task known to work properly.

There can be various types or roles of patterns ([2]), e.g.:

business patterns supporting the interactions of users, business, and data,

architecture patterns supporting the construction of a software architecture – for example integration of several business patterns;

application patterns used to implement interaction of application components and data in business patterns or architecture pattern.

An antipattern is according to [5] a seemingly good solution that is commonly (repeatedly) used but often failing to provide satisfactory results. The specification/description of an antipattern should specify why the antipattern looks to be a recommendable solution. To describe an antipattern it is recommended to specify its symptoms, root causes of the antipattern and consequences of the use of the antipattern [5]. The crucial part of the antipattern description is its (refactored) solution – a way of modifying the antipattern to avoid its wrong consequences; i.e., how to convert it to a good solution.

According to [5] we have in object-oriented environment Software Development Antipatterns, Software Architecture Antipatterns, and Software Project Management Antipatterns. We will see that SOA antipatterns can be of similar types but with different consequences. Some object-oriented antipatterns are patterns in SOA and vice versa (compare the object-oriented antipattern Islands of Automation; it is a crucial SOA pattern). Some SOA solutions solve issues of several problem domains at once (e.g., software development and software management)[1]. The relation between SOA systems and business is tighter than in the object-oriented systems. It leads to yet SOA-specific antipattern: Specification Antipattern referring to wrong attitudes used during the requirements specification phase. Under these circumstances the user involvement in requirement specification as well as in system development [17] is crucial. Users as well as IT professionals should therefore apply some attitudes of agile business and agile software development.

The SOA-related research and experience with SOA in practices produced lists of SOA-related antipatterns [4, 12, 24, 6]. The lists do not attempt to depict the importance of individual antipatterns: how often they occur and how critical losses they cause. Such an evaluation depends, however, on the variant of SOA in which the antipattern is applied. We attempt to evaluate these issues applying the principles of risk management.

[1] All these facts indicate that the service-oriented philosophy is substantially different from the object-oriented one. It can be one of the reasons why SOA is not easy to apply although it seems to be intuitively clear.

2.1. Service-Oriented Software Systems

SOA in our understanding is a virtual peer-to-peer network of software entities called (software) services having many properties common with real-world services. Such systems are formed by the services and a middleware enabling asynchronous communication between the services. The capabilities provided by the middleware vary depending on different conditions. The middleware can include Enterprise Service Bus (ESB, [7]) but the use ESB can be sometimes contraproductive. This issue will be discussed below.

Technically are the services implemented as permanently active service processes communicating asynchronously (batch services are in this case permanently active but having a long latency). We do not exclude systems based on the tools like MessageQueue (MQ) and its descendants. Our concept of service-oriented systems is broader than the one proposed by large software vendors. It is, however, appropriate for many systems supporting, e.g., small-to-medium enterprises or e-government. So the extension of SOA concept we propose is appropriate for many (if not the majority of) systems occurring in practice. For example the integration of existing software items is of the highest priority. It is often denounced not to be good solution. We must understand that the broader treatment of service orientation and service-oriented architecture implies substantially different properties of the resulting systems.

According [15] we can recognize two basic types of service-oriented systems: confederations and alliances. Alliances are collections of components able to search or ask for their cooperating partners. Typical alliances are e-business systems based on web services in the sense of W3C. Confederations are systems where individual components are aware of their cooperating partners. Their communication need not be based on international standards. Typical examples are e-government, information systems supporting global enterprises, or health care systems or some process control (soft real-time) systems.

3. Evaluation of Antipatterns and Risk Management

The research and study of the antipatterns collected a list of (possible) antipatterns occurring in practice. There are risks of loses related to (caused by) each antipattern. From the management's point of view an application of an antipattern usually (with some probability) leads to a project failure. It is therefore a risk. So it is meaningful to apply (adapt) some techniques of risk management [10].

P \ L	low	high
low	low	high
high	low	very high

Table 1. Evaluation of E

According to [10] the risk management consists of the following stages:

1. Risk identification. The construction of the list of antipatterns.
2. Risk assessment. Estimate the expected (i.e., mean) risk loss E(i.e., risk related to the application of an antipattern). It is recommended to estimate E as a product $E = p \cdot L$ where p is the probability that the antipattern will cause the loss L. As we have only rough estimates of p and L, we can use fuzzy estimates (we use the experience of a software developer [27]). It usually suffices to estimate p as *low* and *high* and L as *low*, *high* and set E = *very high* for p = *high* and L = *high*; E = *high* for L = *high* and p = *low*; and E = *low* in other cases. So we have three classes of antipatterns with E = *low*, *high*, and *very high*, see Tab. 1.
3. Risk ordering. We order the antipatterns according the expected loss E. We will discuss the antipatterns having the largest E. It is therefore meaningful to look for the solution of the antipattern having the assessment *very high* (we call such antipatterns critical) and sometimes for the antipatterns with assessment *high*. The assessment should be the part of the description of the antipatterns in a given environment. We shall discuss mainly the critical antipatterns in confederations. Many conclusions hold for all service-oriented systems.

The above assessment can be felt to be rough. In this case we can use for p the degrees *low*, *rather low*, *rather high*, *high*. If necessary, we can extend the scale further. The experience shows that in situations we are discussing the rough assessment is better than the fine one [28]. Sometimes it is good enough to estimate only the level of E directly – without referring to p.

4. What SOA for What Purpose

Simple/small process control systems (real-time systems) were the first systems applying the crucial principles of SOA (see, e.g., [13]). Components (services) were software components driving/supervising rather intelligent devices of real world. Typical example were operation systems of minicomputers and systems controlling manufacturing like flexible manufacturing systems [13] or computer integrated manufacturing (CIM).

The middleware (transport tier) were mainly supported by tools of operating systems like mailbox, the system need not be distributed. The number of components was small and the components were known so the communication protocols and message formats can be agreed. The components were usually written from scratch and they as a rule used interfaces based on a variant of remote procedure call format. The format is "programmer oriented" – i.e., designed mainly to cover the needs of developers.

To summarize – real-time systems have the following features:

- small components developed form scratch,
- mainly known components (we call such SOA *confederations*),
- simple middleware using communication protocols, especially message formats, that need not be user oriented as they are not used by users,
- interfaces tend to be fine grained, procedural, developer oriented,
- not too opened, limited reuse.

Note that communication partners need not be looked for at the start of their dialog.

4.1. Alliances

In e-commerce the communication partners can be looked for all over the world. The implementation of such systems is usually based on web services. It follows that world-wide networks and open standards must be used and the interfaces are difficult to be adapted to specific needs of a given system – it is especially a difficult problem in the situation when immature standards only are available.

Such systems have the following features:

- The communication partners must be looked for at the beginning of communication, possibly all over the world,
- Internet is usually used as (the kernel of) middleware and web services are a good solution,
- almost all the aspects of implementation is based on open standards,
- highly open systems,
- quite frequent use by different users, reuse possible.

We call such systems *alliances* for short [15]. Alliances are typically used in *e*-commerce.

4.2. Confederations

The majority of service oriented systems contain a virtual kernel being a subnetwork of software services providing the basic capabilities of the whole system. The network contains a limited number of well known services. The communication protocols of the services can be agreed. We call such systems *confederations*. Examples are *e*-government, information systems of municipalities, organizations with professional bureaucracy, ERP systems, etc.

The development of confederations strongly depends on the fact whether the service requirement process is supervised by a quite strong central authority or not. The first case is typical for the ERP systems of large enterprises, especially if they have division structure and machine bureaucracy [22].

In the second case the services in the kernel are almost independent, like the states in the European Union. We call therefore such confederations *unions*. It follows from the above list, that software unions occur quite often. They are even typical for the ERP of small firms being often induced to integrate various legacies and third party products.

4.3. Unions

It is crucial to decide what SOA type is to be developed. We can distinguish the following cases:

- *Alliances.* Highly open systems where communicating services must be looked for. They are typical for $e-$commerce.
- *Confederations.* Systems where communicating services in the system kernel knows each other. It is typical for a wide spectrum of SOA systems. There are the following main variants of confederations (Figure 1):
  - *Soft real-time systems.* Almost closed systems with quite small components and typically fine grained interfaces. The interfaces are not intended to be used by users. It is typical for some soft real-time process control systems.
  - *Enterprise confederations.* Semi-open systems supporting large enterprises having machine bureaucracy [23, 22]. The enterprise has enough resources to develop or buy the whole system at once. Architecture is typical for the information systems known as Enterprise Resource Planning (ERP) of global enterprises. Typical is the use of Enterprise Service Bus (ESB, [7]). The core of the confederation is so large that it is meaningful or necessary to use ESB.
  - *Unions.* Almost open SOA systems. Their kernels are built of a quite small number of almost independent services knowing each other. It is the SOA variant typical for the information systems of the large organizations with professional bureaucracy (e-government, schools, etc.) or small or medium-sized enterprises (SME). SME have organization near to ad-hoc-cracy.

The SOA patterns and antipatterns are different for different SOA types. We mainly will discuss the case of unions. Unions rarely use ESB as they integrate a quite small number of applications or systems. Compare the systems of particular offices of a local administration. Unions are typical not only for system having professional bureaucracy but also for also for small and medium-sized enterprises as these enterprises usually do not have enough resources to rebuild their systems completely. Their organization is moreover specific.

In e-government, enterprises (compare [21, 18]), health care systems, etc. the resulting system is built from legacies, third-party products and newly developed systems. It is preferable to wrap the systems such that they have properties mirroring the behavior of real-world services, e.g., asynchronous communication protocols and user-oriented (usable) interfaces. The middleware can and often must use Enterprise Service Bus [7]. It is good when services like data stores and service adapters are used. We call such components *architecture services*.

Unions are formed by a core network of complex application services (mainly legacy systems and third-party products), architecture services, and a middleware. The number of application services is small and the services are known. So their interfaces with the help of architecture services can be agreed – such systems are therefore confederations with large application components connected with help of architecture services and middleware. Some services can communicate with the ”peripheral” or ”foreign” services using the principles used in alliances.

Features of unions in practice:

- large permanently used components that can be integrated into SOA together with their local interfaces (e.g., client tiers) without putting them out of operation for a longer time;
- sophisticated middleware enhanced by architecture services;
- quite large application services – often reused legacy and third-party systems;
- user-oriented coarse-grained interfaces of application services;
- based mainly on the use of open standards but some solutions or their parts (e.g., interfaces) can be propri-

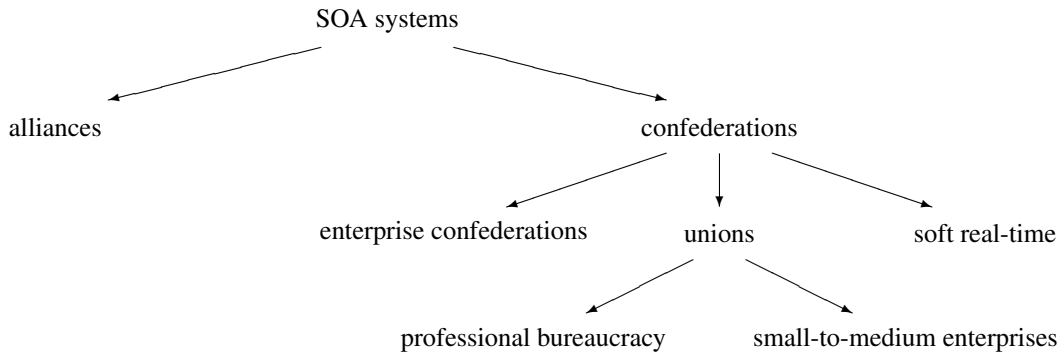


Figure 1. Hierarchy of SOA systems

etary, if appropriate; it is the case when we need the user-oriented declarative coarse-grained interfaces.

- the application services are almost independent (like states in the European Union).

Unions can be used in the development and maintenance of large systems and they can also effectively support the modernization of information systems for small-to-medium enterprises (SME). We will discuss mainly the unions. The reason for it is that we have enough experience with unions and that unions occur frequently. We believe that below given antipattern list is common for all confederations. The evaluation can be different for some antipatterns. Possible candidates for it are "No Legacy", and "Standardization Paralysis".

The situation in alliances is different and will be a topic of further research. In alliances it is more necessary to use open standards, agile business processes are therefore more difficult to apply.

5. Architecture Antipattern: Mis-Selection of SOA Type

SOA is in fact a concept covering multiple technologies. Improper selection of SOA type is therefore an improper selection of a technology to be used.

Symptoms and Consequences

The fact that there are several SOA types, is often ignored. In practice semi-SOA systems are frequently used by wrapping constituent software entities using various message queuing tools like MQ or even low-level tools provided by operating systems.

The different SOA types have different patterns and antipatterns. The proper solutions for different SOA types differ as well. The improper choice or missing choice of proper SOA type leads as a rule to project failure or to substantial losses.

The antipattern is "applied" already in the vision and requirement specification phases – if the requirements are hard to be mapped to possible solutions, or if the used technology restricts the expected features of the system, it is likely that an improper SOA type has been chosen. It can be induced by the application of improper standards or by the marketing of software vendors The typical consequence is failed or too effort consuming a poorly maintainable project often also missing some of the project requirements.

Assessment

The fact that there are several significantly different SOA types is quite unknown. The needed knowledge is blocked by strict standardization effort and by the marketing of large software vendors. p is therefore *high*. The consequences – completely failed project or project fulfilling the goal only partially mean very high loss. L is therefore also *high* to *very high*. Hence E is o *very high*.

Solution

To prevent this antipattern it is good to make the requirements specification precisely and without preselecting the solution type. When the antipattern is already detected, it is necessary to return to the requirements specification and analysis and try to map the requirements to potential solutions. Typical tasks and solutions are discussed below.

Note

It is crucial that SOA in our understanding is any system being a virtual peer-to-peer network of software artifacts behaving like real-world services. This concept is broader than the concepts adopted in SOA related standards, com-

pare the standards by OASIS or W3C or the concept supported by large software vendors.

Too narrow definitions can limit the use of SOA principles and benefit from SOA advantages. It may be one reason of the falling popularity of SOA in the last year, compare the study of Gartner Group from January 2008.

In practice we must integrate batch system, build SOA in bottom up manner etc. It is not difficult to implement it we properly wrap integrated applications and eventually treat them as services communicating in bulk mode and having a great latency. We can occasionally integrate batch applications wrapped by tools like MQ to behave like services.

The techniques developed for service oriented systems can often be used outside SOA. Missing to use it is itself an antipattern. The solution of the discussed antipattern facilitates or implies the solution of several known antipatterns. Examples are: "SOA = Web Services" and "Big Bang" antipatterns.

6. Methodological Antipattern: Fine-Grained Interfaces

Fine-grained interfaces cannot mirror the coarse-grained interfaces of real-world services properly. Fine-grained interfaces have substantial technical drawbacks.

Symptoms and Consequences

People trained to design object-oriented code as a large structured collection of methods and classes being in fact a big collection of procedures each doing almost nothing. It leads in service-oriented framework to the development of systems consisting of services having fine-grained interfaces and being themselves fine-grained. This antipattern was firstly described in [4]. The corresponding communication protocols usually use formats based on the remote procedure call (RPC). Such an attitude is induced also by the fact that RPC is a straightforward way to "reuse" the fine-grained interfaces of software components being in fact wrapped components based on object-oriented methodology.

It all together leads to services being "talkative" and equipped by interfaces that are not user-oriented. The first property leads to the overloading of communication links, the second in fact implies a difficult development of agile business processes, a lot of user discomfort, and a more complex implementation of some management operations like insourcing and outsourcing.

Fine-grained interfaces tend to disclose too much on the technical details of the services. It implies that the interfaces are too much influenced by the changes of the interiors of the services. It is generally known to be an unpleasant property.

Assessment

The antipattern is quite common for all SOA types. p is therefore *high* for all SOA types. The losses L are quite large for unions, especially for the unions for SME. In large enterprises having enough resources to decrease L using appropriate means the problems is not so severe.

For unions p is *high*, L is *low – high*. E is therefore *high – very high*.

For enterprise confederations is L quite *low*. E is in this case *low – high*.

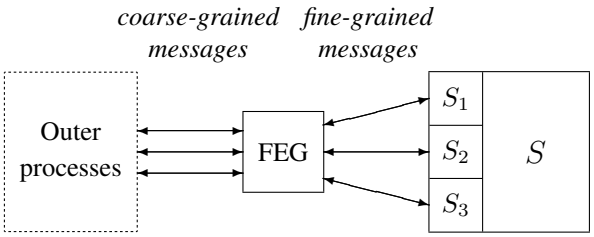


Figure 2. Refactored antipattern Fine-Grained Interfaces

Solution

We can use sophisticated forms of enterprise service bus connectors in the case of enterprise confederations. The second, and may be better, solution is the use of specific architecture services acting as service adaptors. Such services in [20] called *front-end gates* (FEG).

FEG transform n-tuples of fine-grained messages into declarative complex messages having rich semantics, and vice versa. The solution with FEG is applicable for unions as well as for enterprise confederations.

7. Specification and Architecture Antipattern "No Legacy"

Insufficient reuse of legacy systems and third-party products limits substantially the benefits of SOA – especially for small and middle-sized enterprises or for e-government.

Symptoms and Consequences

It is often required that the developed system should not contain any "obsolete" parts – e.g., legacy systems. Compare the antipattern "Lava Flow" [5] known from object-oriented methodology. The reuse of existing software is, however, the most important opportunity of service orientation. In the service-oriented setting it is a very costly antipattern as the main advantage of service orientation is that

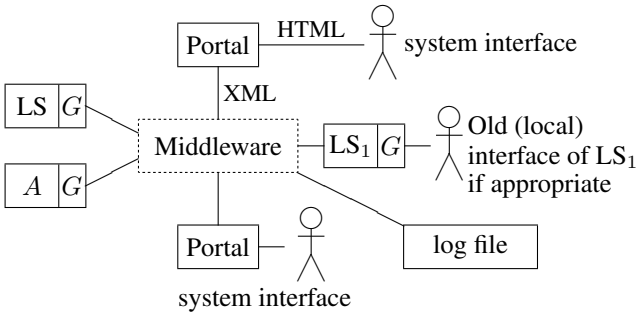


Figure 3. Union integrating legacy systems L-S and LS_1.

it enables the integration of autonomous systems, especially legacy systems. Throwing out of legacy systems causes superfluous immense additional investments into development and implementation of new systems. Very large systems would not be then implementable at all. In other cases it can cause unnecessary testing expenses and retraining of end users.

Assessment

This antipattern occurs very often. The reason is that in the object-oriented world this antipattern is a strongly recommended pattern, compare the object-oriented antipatterns like Island of Automation. Software vendors are not happy that they should integrate and guarantee foreign software. So for unions p is *high* and L is *very high*. The assessment of the antipattern is for unions therefore *very high*.

The consequences of the antipattern bearable for software confederations. In this case p is *high*, L is *high* and E is therefore only *high*.

Solution

At first the project management should accept that it is good to leave some older parts (legacy systems) in a new system as the old parts have useful capabilities and can be very stable. Then we should choose the candidates for integration. The candidates must be then equipped with user-oriented interfaces (portals) to be used by their communication partners. Let us repeat that user-oriented interfaces are based on formalized text straightforwardly mirroring the languages of the users' world. We can use front-end gates [14] to equip the legacy systems with user-oriented interfaces. User-oriented interfaces enables a very powerful implementation of the software engineering principle of information hiding.

If designed carefully, the integration of the systems does not imply any change of "existing" or "old" interfaces of the legacy system LS (Figure 3). Moreover, the local users need not lose their feeling of ownership to "their" system and can bear the responsibility for its data and functions. There can be some political (feeling of power and influence) and organizational (autonomy of divisions/departments or offices in e-government) reasons.

Service orientation is the best way of refactorization of Stovepipe Systems and Stovepipe Factory antipatterns (see [5]). Well working legacy systems simplify outsourcing and reduce the necessity to retrain the end users to work with the new system. Last but not least the use of wrapped legacy systems enables large investment savings – well behaving legacy systems need not be redeveloped, users do less errors and need not be retrained. Note that FEG and portals are from technical or development point of view similar.

Service orientation is the best known way of supporting decentralization of enterprises.

A proper solution of the "No Legacy" antipattern is a precondition of the solution of "Big Bang" antipattern and enables a smooth application of incremental development.

Notes

This antipattern is often a consequence of the antipattern "All From Scratch" and can be also a consequence of the SOA-variant "Vendor Lock-In" antipattern known also from the object-oriented world. In fact the antipattern "No Legacy" is the SOA-variant of the object-oriented antipattern "Reinvent the Wheel" [5].

We have experienced several projects failing due the application of the antipattern "No Legacy". The proposed solution of the antipattern is typically blocked by the antipattern "Standardization Paralysis" attempting to apply cumbersome, complex, and often immature standards everywhere.

8. Management and Design Antipattern "Standardization Paralysis"

The overuse of many (especially immature) standards is contraproductive and can cause some known antipatterns like "Vendor Lock-In" or "Technology Bandwagon".

Symptoms and Consequences

XML enables an easy development of languages transformable very quickly into standards. Many people believe that the standardization based on (quickly changing and obsoleting) standards is a proper solution. The result is that the developers often strongly depend on software vendors and that any solution based on legacies is therefore almost impossible. These effects can be according to [12] known as Technology Altar Antipattern. It often leads to SOA antipatterns like "SOA = Web Services" [4].

The psychological roots of the antipattern has a lot of common with the object-oriented antipatterns "Continuous Obsolence" and "Lava Flow" [5]. An example of this antipattern is the e-government of one country where all the newly incorporated systems must be certified for compatibility with a very quickly changing set of interface rules. It blocks the e-government development.

Assessment

The requirement to standardize all system details is very strong and common. So p is *high*. An improper standardization can induce the use of the antipattern "No Legacy" and limit agile methods of development. It implies the dependency on the product of large software vendors. It implies something like the object-oriented antipattern "Vendor Lock-In". It in fact creates an unnecessary initial barrier for the application of SOA for unions. This consequence is more important for unions where L is therefore *high* to *very high* and E is therefore *high* to *very high* too.

For other SOA types E is *low*.

Solution

In the case of unions and probably in other types of service-oriented systems as well we can at first specify local (private) interface standards, test them for usefulness and other quality aspects in practical applications. Then we can lately transform them into public (ISO) standards after they are available.

9. Specification Antipattern "Business Process for Ever"

It is often believed that the processes are defined so well that no more changes will be necessary. It is in long-term not true even for large enterprises. For smaller companies or institutions in dynamic environment it is not true in almost always.

Symptoms and Consequences

The antipattern can be also called "No Businessmen Involvement" or "Full Automation". It is based on the conviction that well-defined business processes should not be modified easily, if ever. The changes of the process can be implemented by a highly specialized people or teams. It, however, assumes that there are business data of a good quality (i.e., accurate enough, accessible, timely, trustable, not changing quickly, in volumes that are large enough, etc. – compare [29]).

Such assumptions are correct for stable business environments and for large firms having repeatable business processes not too influenced by globalization. Such enterprises have enough resources and enough data to develop processes of a very good quality. It definitely does not hold for small- and medium-sized firms especially in small economies. The books like [9] on the theory of constraints indicate that the assumptions need not hold even for large firms as the business process philosophy must be substantially changed quite often.

If, however, a businessman responsible for the process cannot change the process structure, then he/she cannot be responsible for business consequences of the process. The businessmen cannot even commit business steps if he/she does not know all relevant business information – e.g., trustability of some data that were available to process designer. The situation can be characterized as "fully computerized business processes – no agile user involvement allowed". Such a solution is often unacceptable. Adverse properties of fixed (non-modifiable or modifiable with difficulties) business processes are the following:

1. We can require almost no responsibility of process users (business process owners) for the business process consequences. They can act only as observers and wrong in long term perspectives. It can be fatal in emergency situations.
2. In small and medium firms the data and information are not good enough to enable the definition of stable business processes. This issue is often important for large firms too.
3. The business environment changes, and, e.g., due globalization, the art of business is changing.
4. The detailed specification of business processes is very expensive and time consuming. It cannot be implemented on-line.
5. It is good to train people to cope with unexpected events or changing conditions. We can therefore conclude that it can be often good to allow the users:
   (a) to supervise business processes,
   (b) to commit the business steps if necessary,
   (c) to redefine/change on the fly (in agile style) some parts of the business processes,
   (d) to save/remember the changes of the processes as a part of business intelligence,
   (e) to develop the process from scratch via logging process owner commands.

   Note that a) provides the tools allowing customers and other processes to observe the progress of the process. Similar requirement can be found in [25].

Assessment

There is a widespread conviction that business processes are too important and too difficult to develop to allow business people to change them. The probability of the "use" of the antipattern is therefore *high*. The business consequences tend to be *very high* for unions as agility is there highly desirable. So the assessment of the antipattern is for unions *very high*, i.e., extreme. The antipattern Business Process for Ever is therefore in business environment critical.

The consequences for enterprise confederations are usually bearable (L is *low* to *high*) and as the processes are well designed, p is *low*. We have therefore for this case $E = low$.

Risk evaluation of this antipattern for other SOA types requires further research.

Solution

Any refactoring of the antipattern should enable the on-line user involvement of the user into process execution (it is the responsible user must be able to supervision and on-line modify its processes). The service is defined (e.g., in BPEL [3] or in Aris [11] or UML Action Diagram notation [26]) as a network of actions (operations) provided by software services (we shall call them *application services*). In order to enable the user involvement in business processes it is practically necessary to make application service interfaces user oriented. It is to make the interfaces to mirror user domain professional language, and to mirror (if possible) requests of real-world services (e.g., generate an invoice, bill, consignment/remittance). From the point of view of developer such interfaces are rather declarative (i.e., saying what to do rather than how to do it) and coarse grained.

Using pattern called in [18] *Front-End Gate* (FEG) the interfaces not being user oriented can be usually transformed into user-oriented ones and vice versa.

FEG of an application service S is technically a peer in a peer-to-peer network. Logically it accepts sequences of possibly fine-grained messages from S and transforms them into coarse-grained messages for communication partners of S and vice versa.

The use of FEG can solve the antipattern Chatty Service [4] the assessment of which is *high* to *very high* as L for it is not fatal. FEG can avoid the necessity to use the messages being too developer oriented. It can be called Cyberspace Antipattern. This antipattern usually implies that the service interfaces are not user oriented and it implies the antipattern Business Process for Ever.

The Cyberspace Antipattern occurs quite often, so it should be assessed as critical. FEG can be used to solve antipattern Grey Services when interfaces disclose some implementation details (i.e., services are not used as absolutely black boxes).

Note that user-oriented interfaces simplify the development as they enable the development of powerful screen service prototypes with almost no additional effort [16, 19].

The business services should be implemented such that the implementation fulfills all the conditions a) through e) and additionally enabling the use of business control/modeling data of different types can be based on the pattern Process Manager.

Process Manager

Business processes must admit on-line involvement and supervision of process owners into their execution. According [18] the reasons are:

1. The process model/definition is based on data that need not be timely, accurate, or complete.
2. The business conditions changed or some conditions are not valid any more.
3. The process owner can be obliged to agree with some risky process steps.
4. The information on the process should be understandable for experts (not necessarily IT ones), e.g., at trial.
5. The process model M should be stored as a part of business intelligence.
6. It is desirable to be allowed to have process models in different languages, e.g., in BPEL [3], Aris, [11], workflow [30], or in a semistructured text. The reason is that business process models can be as a part of business intelligence collected during a long time. It is, they must be able to take into account the whole collected experience, e.g., various business documents like manufacturing logs, old business process description documents based on different methodologies. The requirement is especially important for SME where it must be even possible to enable process owners to control business processes having no definition at all or a very informal one. It in fact enables the use of otherwise blocked knowledge of process owners.

As it is not desirable to have much centralized services in peer-to-peer systems (compare experience with UDDI [4]) we can use the following hints:

1. When a process is enacted (typically on the request of its owner O), generate a new service P called *Process Manager*. During the generation a process model M (if any) is transformed into a process control data C parametrizing M using parameters provided by O. O can generate C directly without M, if appropriate. M can be copied from a data store.

2. The process P during its execution generates using C service calls. The calls can be synchronous (call and wait for answer) or asynchronous (just send a message). C can be on-line modified.

3. It is important that if the process owner can supervise the process run and the process run is understandable by non IT experts then the services should have interface based on the languages of user knowledge domains – we say that such interfaces are user oriented. User orientation implies some limitations in the use of classical web services in service-oriented systems. User-oriented interfaces have many software engineering advantages – stability, conciseness, ability to hide implementation details. They enhance reusability of the services. The user-oriented messages are semantically rich, so the communication channels are less loaded.

Using Process Managers to handle business processes is not only refactoring of the antipattern "Business Process for Ever", it is even itself a pattern [20].

10. Design Antipattern "Sand Pile"

The orchestration of lots of small services is sometimes a hopeless issue.

Symptoms and Consequences

This antipattern is also known as "Fine-Grained Services". A frequent implementation of SOA is the technique "one elementary service (e.g., pay-a-bill) per one software component". The result is a large number of small components sharing common data stores. It causes inefficiencies and big maintenance problems. Similar properties are by the antipattern Atomic Services [24] but we think that the crucial problem is a wrong grouping of "atomic" capabilities. It is like to have many highly specialized car service workshops working separately – it is no enterprise providing all repairs at one spot.

Assessment

The antipattern occurs rarely for enterprise confederations as well as for unions (p is *low*) but its consequences are fatal (L = *very high*). So the assessment is in both cases *high*.

Solution

Group related "elementary" services into a composite service with common interface provided by an architecture

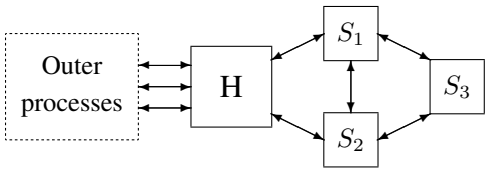


Figure 4. Refactored antipattern Sand Pile

service H called *head of composite service*. For example the composite service can be a collection of services supporting activities of a department. To be more specific: let $S = \{S_1, ..., S_n\}$ be the collection of the services supporting the department. It is required that all the messages address S or from S must pass through H (see Figure 4 and [20]).

11. Antipattern "On-Line Only"

Symptoms and Consequences

Practice indicates that there are frequent situations when some parts of the system must or should be run in batch mode. The main reasons are:

- Some legacy systems are batch systems.
- Some activities have a long latency (as they are computationally complex) or need real-world (e.g., user) responses.
- There can be software engineering reasons to use batch systems (reduction of development effort, security, etc.).

So the integration of batch systems is necessary. The integration can be via data stores implemented as services. There is a prejudice that it is an obsolete technique. The antipattern results into expensive and unstable solutions.

The importance of the combination of batch and on-line applications is discussed in a case study of a flexible manufacturing system [18].

Assessment

The antipattern occurs not too frequently as the integration of batch systems are not frequently used but consequences of it can be very high. So p is *low*, L is *very high*. The assessment E is therefore *high* to *very high* for all confederation types. Note, however, that with growing size of information systems the frequency of the cases when batch systems must be used will grow. Although the pattern may occur also in alliances, risk evaluation of this case has not been done yet.

Solution

It is sometimes good to implement a classical data store but wrap it as a service communicating in bulk mode with batch services being wrapped batch systems. The data store is the filled by batch subsystems in batches and used interactively by other services of the system [18] or vice versa.

The data store can be also used to support the enhancement of communication protocol. In this case the data store contains messages. We call such a data store *message data store service* (MDS). Sometimes a file transfer communication can be used instead data store services.

Note

MDS can be used as a service implementing sophisticated variants of inter-service communication [18]. It can be used for the resolving of, e.g., the antipattern Point-to-Point Service [4] or to implement a very sophisticated communication schemas being more complex than, e.g., publish-subscribe protocol. Data stores and MDS substantially increase the flexibility of SOA using them.

12. Conclusion

The main goal of the research of SOA antipatterns is often aimed on the extension of the list of known antipatterns. The importance of individual antipatterns is seldom discussed.

We have shown that the evaluation the antipatterns has many common features with the stage of risk identification in standard processes of risk management. In fact, there is a risk related to every antipattern. So we should apply consequent stages of risk management for the assessment of antipatterns in order to select and manage the most important risks (antipatterns).

The techniques we have used are very useful as they help to find and properly evaluate the antipatterns being the most important ones from the point of view of risk management. The most critical (meta)antipattern is Mis-Selection of SOA Type.

The further most critical antipatterns are ”Software Processes for Ever” not allowing agile business processes, ”No Legacy”, and ”Standardization Paralysis”. These antipatterns have many links to other SOA antipatterns and even to the antipatterns known from the object-oriented world. In the future we will assess risks of the antipatterns known from the lists mentioned in the references.

Note that the most critical SOA antipatterns are the antipatterns not occurring among the object-oriented antipatterns ([5]). The examples are: ”Mis-Selection of SOA Type” and ”Standardization Paralysis”. Some SOA antipatterns are patterns in object-oriented philosophy (”No Legacy”), and vice versa (”Fine-Grained Interfaces”). The assessment of service-oriented antipatterns has different results for different SOA types. We have discussed some cases of it.

The overall structure of SOA is mainly implied by communication disciplines. They are almost not controlled by any explicit tools, so their use is only the matter of attitude. It substantially increases the flexibility of the SOA systems. If, however, not used properly, it makes the system maintenance a hopeless issue.

Note that the marketing of service-oriented standards have resulted into the situation when there is, according to our meaning, a wrong conviction that SOA requires substantial initial effort and investments. A less dogmatic attitude can allow building systems having no SOA in the strict sense of OASIS and W3C but offering substantial amount of benefits typical for SOA.

It is open whether the differences of unions in small-to-medium enterprises and in large organizations with professional bureaucracy like e-government are not more fundamental than we assumed up to now. It is a very interesting topic for further research.

We applied our evaluation process of antipatterns on the atnipatterns listed in [4, 12]. The results were the following: The evaluation were *low* except the cases when the antipatterns were special instances of the antipatterns from our list. It is important that the majority of the evaluated antipatterns cannot take place or can be solved easily provided that all the antipatterns from our list are solved properly.

Acknowledgement This research was partially supported by the Program ”Information Society” under project 1ET100300517 and by the Grant Agency of Czech Republic under project 201/09/0983.

References

[1] J. Král and M. Žemlička. The most important service-oriented antipatterns. In *International Conference on Software Engineering Advances (ICSEA'07)*, page 29, Los Alamitos, CA, USA, 2007. IEEE Computer Society.

[2] J. Adams, S. Koushik, G. Vasudeva, and G. Galambos. *Patterns for e-Business: A Strategy for Reuse*. MC Press, 2001.

[3] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Specification: Business process execution language for web services version 1.1, 2003. http://www-106.ibm.com/developerworks/library/ws-bpel/ 2009-05-14.

[4] J. Ang, L. Cherbakov, and M. Ibrahim. SOA antipatterns, Nov. 2005. http://www-128.ibm.com /developerworks/webservices/library/ws-antipatterns/ 2009-05-14.

[5] W. J. Brown, R. C. Malveau, H. W. S. McCormick, III, and T. J. Mowbray. *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis*. John Wiley & Sons, New York, 1998.

[6] S. Carter. The top five SOA don'ts, Mar. 2007. http://www.ebizq.net/topics/soa/features/7780.html?related 2009-05-14.

[7] D. A. Chappell. *Enterprise Service Bus*. O'Reilly, 2004.

[8] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison-Wesley, Boston, MA, 1993.

[9] E. M. Goldratt. *Critical Chain*. North River Press, Great Barrington, MA, 1997.

[10] E. M. Hall. *Managing Risks, Methods for Software Systems Development*. Addison Wesley Longman, Reading, MA, USA, 1998.

[11] IDS Scheer. Aris process platform.

[12] S. Jones. SOA anti-patterns, 2006. http://www.infoq.com /articles/SOA-anti-patterns 2009-05-14.

[13] J. Král and J. Demner. Towards reliable real time software. In *Proceedings of IFIP Conference Construction of Quality Software*, pages 1–12, North Holland, 1979.

[14] J. Král and M. Žemlička. Component types in software confederations. In M. H. Hamza, editor, *Applied Informatics*, pages 125–130, Anaheim, 2002. ACTA Press.

[15] J. Král and M. Žemlička. Software confederations and alliances. In *CAiSE'03 Forum: Information Systems for a Connected Society*, Maribor, Slovenia, 2003. University of Maribor Press.

[16] J. Král and M. Žemlička. Service orientation and the quality indicators for software services. In R. Trappl, editor, *Cybernetics and Systems*, volume 2, pages 434–439, Vienna, Austria, 2004. Austrian Society for Cybernetic Studies.

[17] J. Král and M. Žemlička. Systemic of human involvement in information systems. Technical Report 2, Charles University, Faculty of Mathematics and Physics, Department of Software Engineering, Prague, Czech Republic, Feb. 2004.

[18] J. Král and M. Žemlička. Implementation of business processes in service-oriented systems. In *Proceedings of 2005 IEEE International Conference on Services Computing*, volume II, pages 115–122, Los Alamitos, CA, USA, 2005. IEEE Computer Society.

[19] J. Král and M. Žemlička. Software architecture for evolving environment. In K. Kontogiannis, Y. Zou, and M. D. Penta, editors, *Software Technology and Engineering Practice*, pages 49–58, Los Alamitos, CA, USA, 2006. IEEE Computer Society.

[20] J. Král and M. Žemlička. Crucial patterns in service-oriented architecture. In *Proceedings of ICDT 2007 Conference*, page 24, Los Alamitos, CA, USA, 2007. IEEE CS Press.

[21] J. Král and M. Žemlička. Software for small-to-medium enterprises. In M. M. Cruz-Cunha, editor, *Enterprise Information Systems for Business Integration in SMEs: Technological, Organizational and Social Dimensions*. IGI Global, 2009. To appear.

[22] H. Mintzberg. *Mintzberg on Management*. Free Press, 1989.

[23] H. Mintzberg. *Structure in Fives: Designing Effective Organizations*. Prentice Hall, 1992.

[24] T. Modi. SOA antipatterns, Aug. 2006. http://www.ebizq.net /hot_topics/soa/features/7238.html 2009-05-14.

[25] OASIS. Asynchronous service access protocol (ASAP). http://www.oasis-open.org/committees/download.php /14210/wd-asap-spec-02e.doc 2009-05-14.

[26] OMG. Unified modeling language, 2001. Available at http://www.omg.org/technology/documents/formal/uml.htm.

[27] Z. Staníček. Private communication, 2007.

[28] Z. Staníček and J. Hajkr. Project management for implementing is into organizations. In T. Hruška, editor, *DATAKON 2005*, pages 173–197, Brno, Czech Republic, 2005. Masaryk University.

[29] Y. Wand and R. Y. Wang. Anchoring data quality dimensions in ontological foundations. *Commun. ACM*, 39(11):86–95, 1996.

[30] Workflow Management Coalition. Workflow specification, 2004. available at http://www.wfmc.org/standards/docs/Wf-XML-11.pdf.

www.iariajournals.org

International Journal On Advances in Intelligent Systems
ICAS, ACHI, ICCGI, UBICOMM, ADVCOMP, CENTRIC, GEOProcessing, SEMAPRO, BIOSYSCOM, BIOINFO, BIOTECHNO, FUTURE COMPUTING, SERVICE COMPUTATION, COGNITIVE, ADAPTIVE, CONTENT, PATTERNS
issn: 1942-2679

International Journal On Advances in Internet Technology
ICDS, ICIW, CTRQ, UBICOMM, ICSNC, AFIN, INTERNET, AP2PS, EMERGING
issn: 1942-2652

International Journal On Advances in Life Sciences
eTELEMED, eKNOW, eL&mL, BIODIV, BIOENVIRONMENT, BIOGREEN, BIOSYSCOM, BIOINFO, BIOTECHNO
issn: 1942-2660

International Journal On Advances in Networks and Services
ICN, ICNS, ICIW, ICWMC, SENSORCOMM, MESH, CENTRIC, MMEDIA, SERVICE COMPUTATION
issn: 1942-2644

International Journal On Advances in Security
ICQNM, SECURWARE, MESH, DEPEND, INTERNET, CYBERLAWS
issn: 1942-2636

International Journal On Advances in Software
ICSEA, ICCGI, ADVCOMP, GEOProcessing, DBKDA, INTENSIVE, VALID, SIMUL, FUTURE COMPUTING, SERVICE COMPUTATION, COGNITIVE, ADAPTIVE, CONTENT, PATTERNS
issn: 1942-2628

International Journal On Advances in Systems and Measurements
ICQNM, ICONS, ICIMP, SENSORCOMM, CENICS, VALID, SIMUL
issn: 1942-261x

International Journal On Advances in Telecommunications
AICT, ICDT, ICWMC, ICSNC, CTRQ, SPACOMM, MMEDIA
issn: 1942-2601