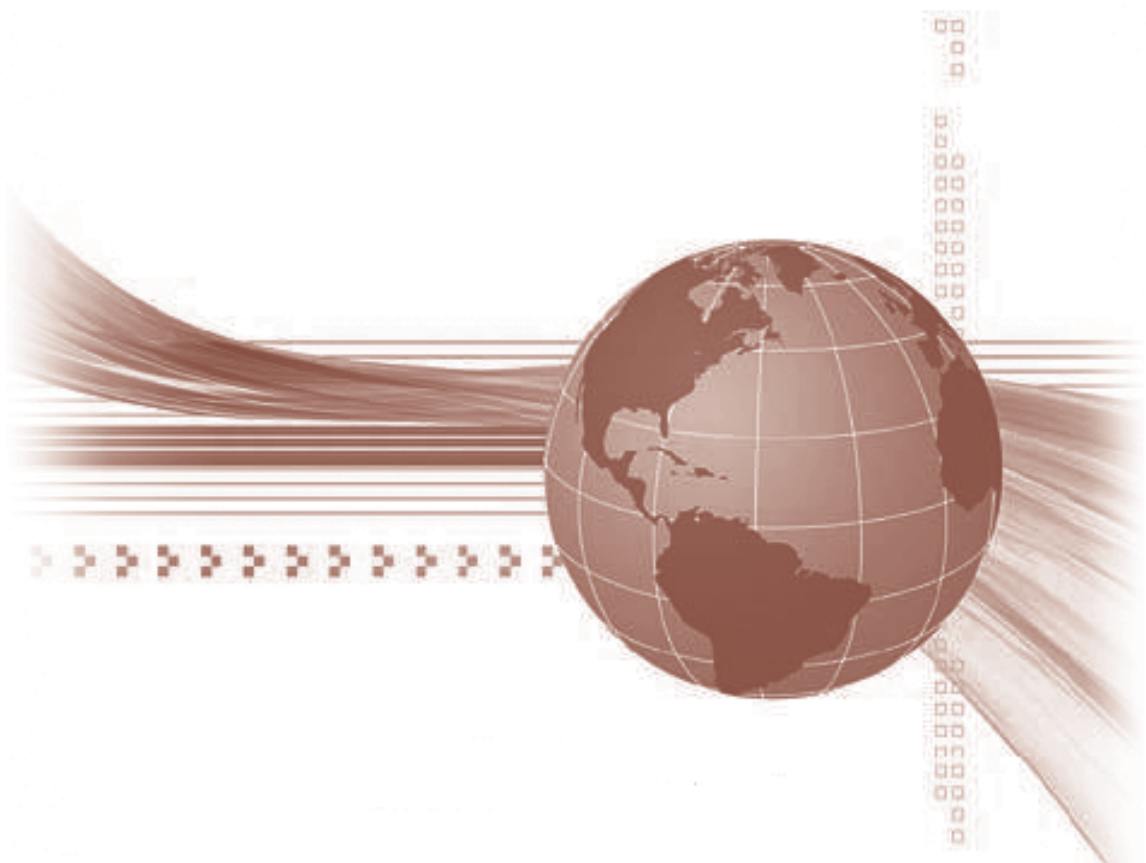
STUDIA UNIVERSITATIS
BABEŞ-BOLYAI



INFORMATICA

1/2012

YEAR **Volume 57 (LVII) 2012**
MONTH **MARCH**
ISSUE **1**

S T U D I A
UNIVERSITATIS BABEŞ-BOLYAI
INFORMATICA

1

EDITORIAL OFFICE: M. Kogălniceanu 1 • 400084 Cluj-Napoca • Tel: 0264.405300

SUMAR – CONTENTS – SOMMAIRE

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVII**, Number 1, 2012

A MEMBRANE-IMMUNE ALGORITHM FOR SOLVING THE MULTIPLE 0/1 KNAPSACK PROBLEM

EMAD NABIL[1], AMR BADR[2], AND IBRAHIM FARAG[2]

Abstract. In this paper a membrane-immune algorithm is proposed, which is inspired from the structure of living cells and the vertebrate immune system. The algorithm is used to solve one of the most famous combinatorial NP-complete problems, namely the Multiple Zero/One Knapsack Problem. Various heuristics, like genetic algorithms, have been devised to solve this class of combinatorial problems. The proposed algorithm is compared with two genetic based algorithms and overcame both of them. The algorithm is evaluated on nine benchmarks test problems and surpassed both of the genetic based algorithms in six problems, equaled with one of them in two problems and lost in one problem, which indicates that our algorithm surpasses in general genetic algorithms. We claim that the proposed algorithm is very useful in solving similar combinatorial NP-complete problems.

1. Introduction

Membrane computing is a branch of natural computing which investigates computing models abstracted from the structure and the functioning of living cells and from their interactions in tissues or higher order biological structures.

Membrane Computing was introduced by Gh. Paun in [6] under the assumption that the processes taking place in the compartmental structure of a living cell can be interpreted as computations. The devices of this model are called P systems. A P system consists of a membrane structure, in the compartments of which one places multi-sets of objects which evolve according to given rules in a synchronous non-deterministic maximally parallel manner. The basic idea is to consider a distributed and parallel computing device structured in an arrangement of membranes which delimit compartments

Received by the editors: October 12, 2011.

2010 *Mathematics Subject Classification.* 68T20, 92D25, 92F05.

1998 *CR Categories and Descriptors.* I.2.8 [**Computing Methodologies**]: artificial intelligence – *Problem Solving, Control Methods, and Search*; I.1.2 [**Computing Methodologies**]: Symbolic and algebraic manipulation – *Algorithms*.

Key words and phrases. membrane computing, P systems, artificial immune system, clonal selection algorithm, Knapsack Problem.

where various chemicals (objects in our case) evolve according to local reaction rules. The objects can be eventually sent to the environment or to adjacent membranes under the control of specific rules. The reaction rules are applied in a parallel manner, with the objects to evolve by them and with the reactions themselves chosen in a non-deterministic manner. In this way, we can define transitions from one configuration to another configuration of our system and hence we can define computations [7]. A computation provides a result, for instance, in the form of the number of objects present in the halting configuration in a specified compartment, or in the form of a special object, yes or no, sent to the environment at the end of the computation, this is the way of answering a decision problem that the system had to solve. An abstraction of membrane structure is depicted in figure 1.

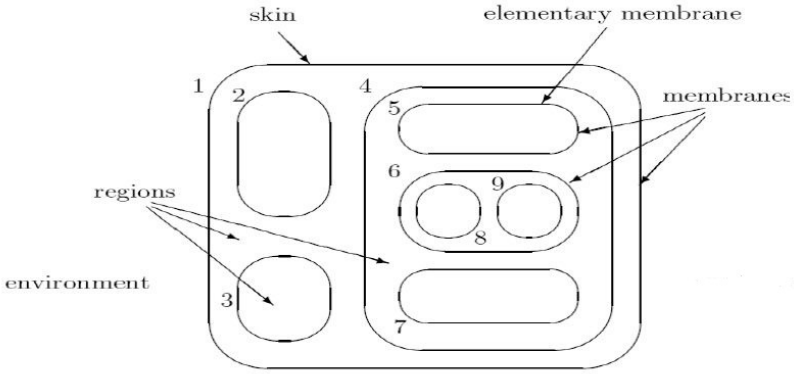


FIGURE 1. the membrane structure.

This paper suggests a membrane-immune algorithm which is an approximate algorithm for NP-complete optimization problems, the membrane-immune algorithm borrows nested membrane structures, rules in membrane separated regions, transporting mechanisms through membranes from P systems and the cloning selection principle which is imported from the vertebrate immune system and uses all these ingredients in solving NP-complete optimization problems approximately. We evaluated the proposed algorithm by applying it to one of the most famous NP-complete problems, the multiple 0/1 knapsack problem. The paper is organized as follows. Section 2 describes the multiple 0/1 knapsack problem, in section 3 an explanation of the clonal selection

principle is given, section 4 explains in detail the proposed algorithm and the experimental results, finally conclusions and some remarks are discussed in section 5.

2. The Multiple Zero/One Knapsack Problem

The problem we study here is a generalization of the 0/1 simple knapsack problem. In the simple version, we are given a knapsack of capacity c, and n objects. Each object has a weight w_i, and a profit p_i, $1 \leq i \leq n$, The objective is filling the knapsack with the objects that yield the maximum profit without firing the capacity constraint. The problem is also known as the single-line integer programming problem [3, 4].

The multiple 0/1 knapsack problem consists of m knapsacks of capacities c_1, c_2,...,c_m and n objects, each of which has a profit p_i, $1 \leq i \leq n$. Unlike the simple version in which the weights of the objects are fixed, the weight of the ith object in the multiple knapsack problem takes j values, $1 \leq j \leq m$. The ith object weighs w_{ij} when it is considered for possible inclusion in the jth knapsack of capacity c_j. The objective is finding a vector $\vec{x}$=(x_1, x_2,..., x_n) that guarantees that profit is maximized and in the same time takes into consideration that no knapsack is overfilled.

This problem is also known as the zero/one integer programming problem [12], and as the 0/1 linear Programming problem [14]. The multiple 0/1 knapsack problem can be thought as a resource allocation problem, where we have m resources (the knapsacks) and n objects. Each resource has its own budget (knapsack capacity), and w_{ij} represents the consumption of resource j by object i. Once more, we are interested in maximizing the profit, while working within a certain budget.

The popularity of knapsack problems stems from the fact that it has attracted theoreticians as well as practitioners [14]. Theoreticians enjoy the fact that these simple structured problems can be used as sub-problems to solve more complicated ones, practitioners on the other hand, enjoy the fact that these problems can model many industrial opportunities such as cutting stock, cargo loading, and the capital budget. Table 1 gives a formal definition of the multiple 0/1 knapsack problem.

2.1. **Related Work.**

P systems are parallel molecular computing models based on processing multisets of objects in cell-like membrane structures. In [9] a membrane algorithm used to solve the multidimensional 0-1 knapsack problem in linear time is given. The algorithm uses P system recognizers with input and with active membranes using two-divisions. This algorithm can also be modified to solve general 0-1 integer programming problem. In [11] the Knapsack problem is solved using a family of deterministic P systems with

active membranes using two-divisions. The number of steps of any computation is of linear order, but a polynomial time is required for pre-computing resources.

The works in [9, 11] use membrane computing to generate the whole exponential search space, then apply an exhaustive search to find the global optimal solution. Until now membrane algorithms have no real implementation and still in its infancy form. Researchers till the moment do not have a precise conception about implementation of p systems, an in vitro, in vivo or in silico implementation. Our algorithm differs from the two works previously mentioned that it produces an approximate accepted solutions and can be implemented in the current silicon based computers, and easily can run using parallel or grid computing.

TABLE 1. The multiple 0/1 knapsack problem formalization

knapsacks:	$1, 2, ..., m$
capacities:	$c_1, c_2, ..., c_m$
objects:	$1, 2, ..., n$
profits:	$P_1, P_2, ..., P_n$
Weights:	$KS_1 : w_{11}, w_{12}, ..., w_{1n}$ $KS_2 : w_{21}, w_{22}, ..., w_{2n}$ ..... $KS_m : w_{m1}, w_{m2}, ..., w_{mn}$
Feasible solution:	$\vec{x}$=(x_1,..., x_n), $x_i \in \{0, 1\}$, s.t. $\sum_{i=1}^{n} w_{ij}x_i \leq c_j, 1 \leq j \leq m$
Affinity function:	Maximize $\sum_{i=1}^{n} P_i x_i$
Optimal solution:	A feasible solution that gives the maximum profit

3. The Clonal selection principle

The clonal selection principle is an algorithm used by the immune system to describe the basic features of an immune response to an antigenic stimulus. An abstraction of clonal selection principle [10] is described in figure 2.

The principle establishes the idea that only those cells that recognize the antigens are those with high ratio proliferate. Clonal selection operates on both T cells and B cells. The immune response occurs inside the lymph nodes. When an animal is exposed to an antigen, some subpopulation of its bone marrow's derived cells (*B lymphocytes*) respond by producing antibodies. Each cell secretes only one kind of antibody, which is relatively specific for the antigen. By binding to these immune receptors, with a second signal from accessory cells, such as the T-helper cell, an antigen stimulates the B cell to proliferate (divide) and mature into terminal (non-dividing) antibody

secreting cells, called plasma cells. While plasma cells are the most active antibody secretors, large B lymphocytes, which divide rapidly, also secrete Ab, albeit at a lower rate. While B cells secrete Ab, T cells do not secrete antibodies, but play a central role in the regulation of the B cell response and are core in cell mediated immune responses. Lymphocytes, in addition to proliferating or differentiating into plasma cells, can differentiate into long-lived B memory cells. Memory cells circulate through the blood, lymph and tissues, probably not manufacturing antibodies, but when exposed to a second antigenic stimulus commence differentiating into large lymphocytes capable of producing high affinity antibody, preselected for the specific antigen that had stimulated the primary response. The main features of the clonal selection theory [5] are:

- The new cells are copies of their parents (clone) subjected to a mutation mechanism with high rates (*somatic hypermutation*);
- Elimination of newly differentiated lymphocytes carrying self-reactive receptors;
- Proliferation and differentiation on contact of mature cells with antigens;
- The persistence of forbidden clones, resistant to early elimination by self-antigens, as the basis of autoimmune diseases.

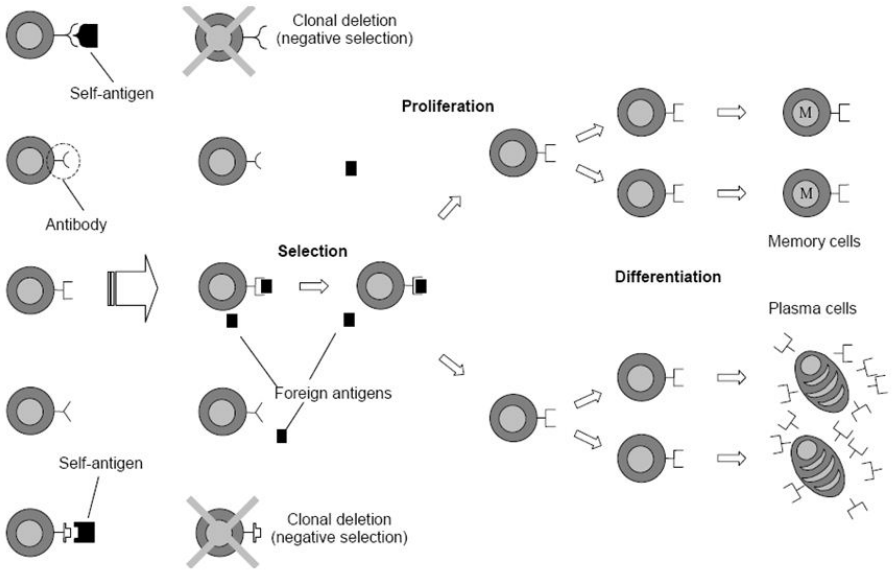


FIGURE 2. The clonal selection principle.

The fittest clones are the ones that best recognize antigens or, more precisely, the ones that are triggered best. For this algorithm to work, the receptor population or repertoire has to be diverse enough to recognize any foreign shape. A mammalian immune system contains a heterogeneous repertoire of approximately 10^{12} lymphocytes in human, and a resting (unstimulated) B cell may display around $10^5 - 10^7$ identical antibody-like receptors. The repertoire is believed to be complete, which means that it can recognize any shape of an antigen [10].

4. The Membrane-Immune Algorithm and Experimental Results

The membrane-immune algorithm depicted in figure 3 is described below as Pseudocode.

```
1. Begin
2.   Initialize the initial repertoire R randomly;
3.   Identify the affinity measure;
4.   Validate repertoire;
5.   While (condition) do
6.     Begin
7.        For each P system in R.
8.        Begin
9.         a. Perform mutation over all solutions in each compartment;
10.        b. In every region, the best and worst solutions are sent
              to the adjacent inner and outer regions, respectively;
11.       End;
12.      Perform Clonal/Negative Selection according affinity;
13.      Metadymanics;
14. End;
15. End;
```

In the following subsections an explanation of how the algorithm is working in detail is given.

4.1. **The Algorithm Structure.**

The repertoire here is a set of P systems, each one consists of set of nested membranes, each region has a set of solutions and a simple mutation algorithm that mutates all solutions in this region, as depicted in figure 4. These solutions are initialized randomly.

In every region there are a few solutions of the optimization problem to be solved and a mutation algorithm works on them. After the initial settings all solutions are updated by the mutation algorithm placed in regions, simultaneously, in every region, the best and worst solutions, with respect to the optimization criterion, are sent to the adjacent inner and outer regions, respectively. The best solution exists in the innermost region of a P system.

The process of updating and transporting solutions is repeated until a termination condition is satisfied. In our implementation a fixed number of

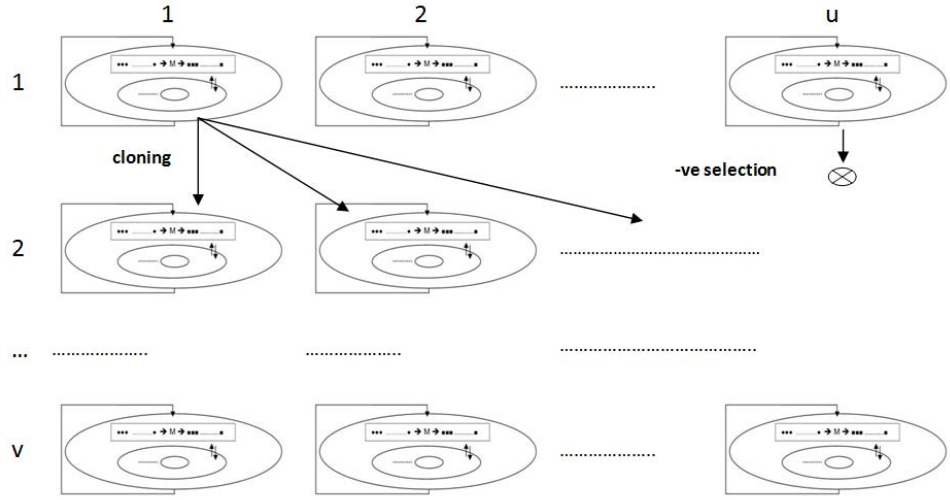


FIGURE 3. How the membrane-immune algorithm works, u represents the repertoire size; v represents how many times the cloning selection principle is applied.

iterations is used as a termination condition as explained in the algorithm setting, see subsection 4.8.

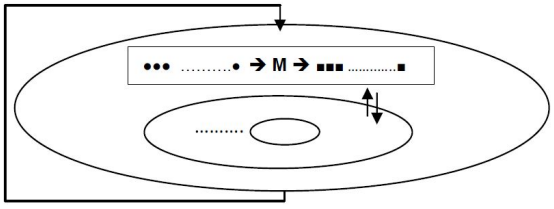


FIGURE 4. The Structure of repertoire's single p system.

4.2. **Repertoire Initialization.** All generated p systems' initial solutions are feasible. This means that all the generated solutions satisfy the knapsack constraints, and all subsequently infeasible individuals were made feasible by dropping some items randomly from the knapsacks until feasibility is obtained.

It might be allowed to let a p system to contain infeasible individuals (representing invalid solutions), to increase the variation of solutions. The infeasible individuals are often penalized so as to means lower affinity when

comparing to other feasible solution. According to [1] it does not indicate that it gives any benefit to allow infeasible solutions to be included, so we did not permit infeasibility of solutions in our implementation.

4.3. **The affinity measure.** The affinity measure for a solution is the same as described in table 1.

4.4. **Repertoire Validation.** As mentioned above infeasible solutions are not included in the repertoire, and thus no penalty function is used. Due to random initialization or mutation, infeasible solutions are produced, solutions' validation is performed by choosing a random bit contains value 1 to be flipped into 0 until this solution satisfies all knapsacks' constraints.

4.5. **Cloning Selection Mechanism.** Each p system in the repertoire is supposed to be enhanced continuously by time because of maturation performed by mutation. The higher affinity solutions will be found in the inner most regions and cloning will performed proportional to these solutions' affinity, i.e. a high membrane inner most solution affinity means a high cloning rate; a low membrane inner most solution affinity means a low cloning rate or a negative selection (clonal deletion).

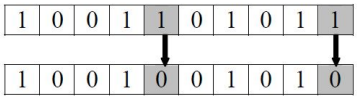


FIGURE 5. An example of mutation.

4.6. **Mutation.** The concept of mutation is essential to membrane-immune algorithm, as it is the only way for repertoire diversification and maturation. Without sufficient mutation the population will tend to converge towards a few good solutions, possibly representing local optima. On the other hand, with too much mutation the search will have problems of focusing on potentially good solutions; this is the tradeoff between exploration and exploitation. According to our experiments we found that 0.1 mutation rate is suitable.

4.7. **Meta-dynamics.** To keep the repertoire diversity, a number of randomly generated P systems are added to the repertoire, our experiments showed that there is no need to perform this step, but in other problems it could be useful.

4.8. **Parameters Setting.** The following setting is used in our experiments: Repertoire size = 30 P systems, maximum iterations number of each P system = 50, each P system in the repertoire has 100 nested membranes, proportional cloning is used according to the highest inner most solution affinity, clonal/negative selection iterations is performed 100 times. The proposed algorithm is tested on 9 benchmarks problem and every problem is solved 100 times. the 30 P systems that compose the population are independent at the iteration level, so they can work in a destribued system for effeciency measures.

Parameters setting in work mentioned in [2] are as follows, the population size is 100 individual, cross over rate=0.9, mutation rate = 0.01, No cloning is applied. The algorithm uses proportional selection. Total number of 100 runs is executed for each test problem. parameters setting in work mentioned in [13] are as follows, population size = 100, crossover rate = 0.6, mutation rate = 1/n, where n is the population size, No cloning is applied, The algorithm uses proportional selection. Total number of 100 runs is executed for each test problem.

Optimal solution in table 3 means the known optimum. The known optimum may be enhanced using the following suggestions, using a different stopping condition rather than fixed number of iterations, i.e. a heuristic termination condition, for example the algorithm stops when there no enhancement or the enhancement is less than certain value, also the mutation value could be degraded rather than fixed value, i.e. in the first iterations the mutation is high and degrades by incrementing iterations, this is under the assumption that the whole individuals are enhanced by time. Increasing the number of P systems allows the coverage of more search space and this could result in general enhancement to reach better optimal.

The membrane-immune algorithm is evaluated on the same nine test problems, for comparison reasons, which are previously solved by [2, 13], the abbreviations CT will be used to refer to the work [2] , and KBH for the work [13], while NBF will be used for referring to this paper. The problem sizes range from 15 to 105 objects and from 2 to 30 knapsacks. A collection of all of these problems is available from the OR-library by Beasley [8]. All information about each problem is explained in table 2, number of knapsacks, number of objects, optimal solution, also average value of solutions found by the membrane-immune algorithm in comparison with KBH and CT. Table 3 and figure 6 explains the degree of how much each algorithm finds optimal solution.

The membrane immune algorithm with the previously mentioned setting in subsection 4.8. is able to find the global optimum point exactly for all problems with a clear difference from KBH and CT, but the exception is given by the problem weing7-105. the previous setting worked badly, so we

changed the repertoire size to be 100, and the number of cloning iterations to be 400. Nevertheless most of the runs get stuck before finding the optimal solution, but still the average value of the found solutions is accepted.

The percent of optimum solutions found and the average affinity value over all runs turns out to results indicates that the membrane-immune algorithm can serve as a fast and robust approximation heuristic for combinatorial problems.

TABLE 2. Experimental results of KNH, CT and NBF that represent the average value of execution using the nine benchmarks problems solved by Khuri *et al.* in [13] and Cotta, Troya in [2].

Problem	Knapsacks	Objects	Optimal	Average			
				KBH	CT	NBF	Winner
Knap15	10	15	4015	4012.7	4015.0	4015.0	CT, NBF
Knap20	10	20	6120	6102.3	6119.4	6120.0	NBF
Knap28	10	28	12400	12374.7	12400.0	12400.0	CT, NBF
Knap39	10	39	10618	10546.9	10609.8	10618.0	NBF
Knap50	10	50	16537	16378.0	16512.0	16528.3	NBF
Sento1	30	60	7772	7626	7767.9	7772.0	NBF
Sento2	30	60	8722	8685	8716.5	8721.95	NBF
Weing7	2	105	1095445	1093897	1095386.0	1094775.0	CT
Weing8	2	105	624319	613383	622048.1	624319.0	NBF

TABLE 3. A Comparison between KNH, CT and NBF explains how much each algorithm finds the optimal solution

Problem	KBH	CT	NBF
Knap15	83%	100%	100%
Knap20	33%	94%	100%
Knap28	33%	100%	100%
Knap39	4%	60%	100%
Knap50	1%	46%	50%
Sento1	5%	75%	100%
Sento2	2%	39%	95%
Weing7	0%	40%	5%
Weing8	6%	29%	100%

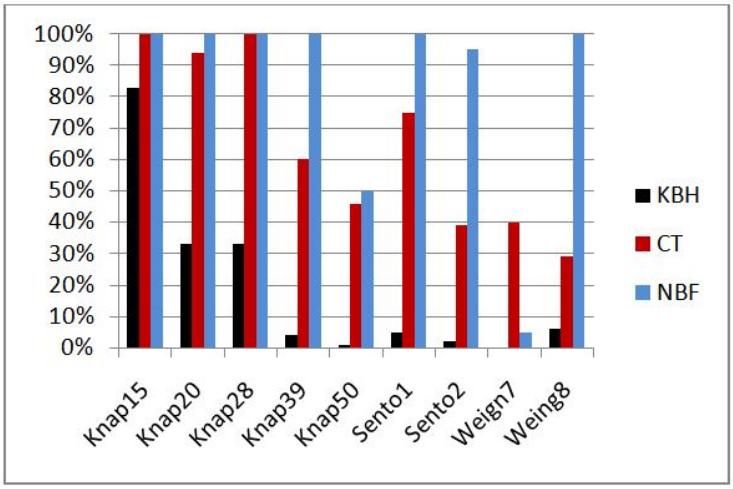


FIGURE 6. A Comparison of the three algorithms explains the percent of finding the optimal solution.

5. Conclusions and Remarks

We have proposed the membrane-immune algorithm which is inspired from the structure of living cells and the vertebrate immune system. The algorithm is evaluated by solving a famous NP-complete optimization problem namely the multiple 0/1 knapsack problem and overcame two genetic based algorithms.

Although there is a great success in the design of theoretical solutions to NP problems, these solutions have a fundamental drawback from a practical point of view. It is not clear yet what is the actual real implementation of Membrane Computing, an *in vitro, in vivo* or *in silico* implementation, any case, a membrane will have a space associated that could be a piece of memory in a computer, a pipe in a lab or the volume of a bacteria, only brute force algorithms will be able to implement little instances of such problems. If we take an *in vivo* implementation where each feasible solution will be encoded in an elementary membrane and such elementary membrane is implemented in a bacteria of mass $\cong 7 \times 10^{-16}$ kg., for example E. Coli, then, a brute force algorithm which solves an instance of an NP problem with an input size, for example 40 will need approximately a mass $\cong 6 \times 10^{24}$ kg., which equals approximately mass of the earth, so it is not practical for a P system to be implemented in solving relatively large NP-Complete problems, the proposed

membrane-immune algorithm can find approximate accepted solutions and still has the feasibility for a practical real implementation in terms of resources.

There are many possibilities for improving membrane-immune algorithm, for example different termination conditions could be used, i.e. one can terminate execution if the good solution is not changed during a predetermined number of steps, considering meta-dynamics in the earlier execution in the algorithm could enhance performance, meta-dynamics may be useful in some applications and in others not, also instead of performing cloning according to the affinity of the inner most solution, it could be performed according to the average value of all solutions for a P system, the first mechanism takes into consideration the affinity of one solution to perform cloning but the latter counts for all solutions' affinity in a p system. Because the repertoire has a number of independent P systems, the algorithm will be easily implemented in parallel, distributed, or grid computing systems which indicates a better performance. Each p system itself could be implemented in a parallel system as it contains a number of independent regions all of which has its solutions and maturation/mutation technique. Also different mutation techniques may be more suitable for different NP-complete problems like mutation per solution or Swap Mutation; the best mutation rate for repertoire is an open point, mutation may be high in early iteration and get smaller by time. The proposed structure could be involved in more complex structures as well.

Our positive results support the idea that membrane-immune algorithm is a suitable approach for tackling highly constrained NP-complete problems.

References

[1] A. Hoff, A. Lokketangen, I. Mittet, *Genetic Algorithms for 0/1 Multidimensional Knapsack Problems*, Proceedings of Norsk informatikk konferanse, NIK'96, pp. 291–301, 1996.

[2] C. Cotta, J. M. Troya, *A hybrid genetic algorithm for the 0-1 multiple knapsack problem*, In: G.D. Smith, N.C. Steele and R.F. Albrecht (eds.), *Artificial Neural Nets and Genetic Algorithms* Vol. 3., Springer-Verlag, Wien New York, 1998, pp. 251–255.

[3] C. H. Papadimitriou, K. Steiglitz, *Combinatorial Optimization:Algorithms and Complexity*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1982.

[4] D. R. Stinson, *An Introduction to the Design and Analysis ofAlgorithms*, The Charles abbage ResearchCenter,Winnipeg, Manitoba, Canada, 2nd edition, 1987.

[5] F. M. Burnet, G. I. Bell, A. S. Perelson, Jr. G. H. Pimbley, *Clonal Selection and After*, In Theoretical Immunology,(Eds.), New York: Marcel Dekker Inc., 1978, pp. 63–85.

[6] G. Paun, *Computing with membranes*, TUCS Report 208, Turku Center for Computer Science, 1998.

[7] G. Paun, *Membrane Computing. An Introduction*, Springer-Verlag, Berlin, 2002.

[8] J. E. Beasley, OR-Library: Distributing TestProblems by Electronic Mail, *Journal of Operational Research Society*, Vol. 41, No. 11,1990 ,pp. 1069–1072

[9] L. Pan, C. Martin-Vide, *Solving Multiset 0-1 Knapsack Problem by P Systems with Input and Active Membranes*. In Proceedings of the Second Brainstorming Week on Membrane

Computing, Sevilla, Spain, BWMC 2004, Report RGNC 01/04, University of Sevilla, 2004, pp. 342–353.

[10] L.N. De Castro, J.I. Timmis, *Artificial immune systems: A new computational intelligence approach*, Springer-Verlag, 2002.

[11] M. J. Perez-Jimenez , A. Riscos-Nunez, *A Linear-Time Solution to the Knapsack Problem Using P Systems with Active Membranes*. Membrane Computing, 2004, pp. 250–268.

[12] M. R. Garey, D. S. Johnson, *Computers and Intractability:A Guide to the Theory of NP-Completeness*, W. H. Freemanand Company, San Fransisco, 1979.

[13] S. Khuri, T. Back, J. Heitkotter, *The zero/one multiple knapsack problem and genetic algorithms*, Proceedings of the ACM Symposium on *Applied Computing*, SAC '92, ACM Press, pp.188–193, 1994.

[14] S. Martello, P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*, John Wiley & Sons, Chichester,West Sussex, England, 1990.

[1]Department of Computer Science, Faculty of Information Technology, Misr University for Science and Technology, Al-Motamayez District, Postal code: 15525, 6th of October City, Egypt

[2] Department of Computer Science, Faculty of Computers and Information, Cairo University, 5 Dr. Ahmed Zewail Street, Postal Code: 12613,Orman, Giza, Egypt

E-mail address: emadnabilcs@gmail.com

E-mail address: a.badr.fci@gmail.com

E-mail address: i.farag@fci-cu.edu.eg

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVII**, Number 1, 2012

SPECIAL ATTRIBUTES FOR DATABASE NORMAL FORMS DETERMINATION

VITALIE COTELEA

Abstract. The article deals with the relational schemes defined on the reduced sets of functional dependencies divided into equivalence classes. The concepts of non-essential and recoverable attributes are introduced, and the algorithms for their computing are proposed. Some properties of schemes, based on these attributes, are presented as well.
Based on non-essential and recoverable attributes some conditions are introduced to make a relation be in the third or Boyce-Codd normal form. These conditions are just sufficient, because there could be a scheme that is in third or Boyce-Codd normal form but doesn't satisfy them. In addition, the article suggests a normalization algorithm that takes into account these conditions.

1. Introduction

A category of problems that may arise in the development of applications using a database is the incorrect designed relational schemes. Testing the correctness of a scheme can be made using functional (or other) dependencies attached to that scheme.

Many decision-making problems related to functional dependence schemes are difficult to calculate. Such issues include determining if an attribute is primary and testing if a scheme is in a certain normal form. Algorithms for these problems are required in design tools for databases. But these problems can be solved only by algorithms of exponential complexity for the time being. Even for schemes with a relatively small number of attributes, such algorithms can not be used for all design tasks.

Testing if a relational scheme is in Boyce-Codd normal form is an easy problem. In fact, it must be tested for all dependencies defined on the scheme, if their left sides are super keys [11]. This clearly is made in polynomial time,

Received by the editors: January 10, 2012.

2010 *Mathematics Subject Classification.* 68P05, 68P15.

1998 *CR Categories and Descriptors.* H.2.1 [**Database Management**]: Logical Design – *Normal forms* .

Key words and phrases. Database, Logical design, Non-essential attributes, Recoverable attributes, Third normal form, Boyce-Codd normal form.

in contrast to testing if the scheme is in third normal form, which is a NP-complete task, as testing the primeness of attributes is NP-complete [8].

It should be mentioned that the first statement is correct, only if the given set of dependencies is an exhaustive one. In case it does not represent the closure of the set of dependencies, the problem is of an exponential character. Although the Boyce-Codd normal form testing if the above condition is satisfied, is executed in polynomial time, to detect whether a subscheme of a scheme is in this form, is an NP-complete problem [2]. The reason of the increasing complexity lies in the following: For the Boyce-Codd problem the relational scheme is given. In other words, the set of functional dependencies is part of the input. And the only remaining thing is to test if the left side of each dependency is a superkey. But in the case of a subscheme, the set of dependencies is not known explicitly.

Worland [13] presents an algorithm that determines whether a scheme is in third normal form. The algorithm works, classifying scheme attributes in so-called dependent sets, which are based on the set of functional dependencies defined on the given scheme. To visualize the dependencies, a new type of dependency graph is introduced. The algorithm works faster than the designed algorithms for finding all candidate keys of the scheme, especially if there is more than one dependent set.

Obviously the question arises if recognizing the schema's normal form has a pre-requisite to determine the prime and nonprime attributes or finding of all keys. A solution would be to determine the equivalent characteristics of these entities, in such a way that it could be calculated in polynomial time.

This article introduces the notion of two types of attributes: essential and recoverable and it gives some of their properties that can be used to analyze database schemas. The algorithms for determining the essential and recoverable attributes are likely polynomial.

2. Preliminary notions

This section presents some known concepts from where we will proceed to describe the subject of this paper.

Let $Sch = (R, F)$ be a database schema, where F is a set of functional dependencies over a set R of attributes. The set F of functional dependencies may have a diverse structure. First, it can be a nonredundant set, secondly, it may be a reduced set, divided into equivalence classes or may be a minimum set [9, pp.71-85].

Two sets of functional dependencies F and G are equivalent (given as $F \equiv G$), if and only if $F^+ = G^+$, that is, if the closures of these sets are equal.

A set F of functional dependencies is nonredundant [12], if $\nexists G$, so that $G \subset F$ and $G \equiv F$.

Consider marking by $|F|$ and $||F||$ the cardinality of F and the number of attributes involved by F respectively (including the repeating ones).

The set F of functional dependencies is a minimum set [10], if $\nexists G$, such that $G \equiv F$ and $|G| < |F|$.

Let F be a set of functional dependencies over a scheme R and $X \to Y \in F$. The attribute A is extraneous in dependency $X \to Y$ with respect to F, if

$$A \in X \text{ and } F - \{X \to Y\} \cup \{(X - \{A\}) \to Y\} \equiv F$$

or

$$A \in Y \text{ and } F - \{X \to Y\} \cup \{X \to (Y - \{A\})\} \equiv F.$$

A set F of functional dependencies is left-reduced (right-reduced), if every functional dependency in F has no redundant attribute on the left (right) side. A left and right-reduced set of functional dependencies is a reduced set.

Let $X \to Y \in F$. It is defined as the equivalence class of functional dependencies which includes $X \to Y$ the set of dependencies, written $E_F(X)$:

$$E_F(X) = \{V \to W | V \to W \in F \wedge X \leftrightarrow V\}$$

So, $E_F(X)$ is the set of functional dependencies in F with left sides equivalent to X with respect to F.

The set of attributes $K \subseteq R_i$ is a key of a scheme $Sch_i = (R_i, F)$, if it satisfies the following conditions:

(1) $K \to R_i \in F^+$
(2) $\forall K' \subset K,\ K' \to R_i \notin F^+$.

The set $K \subseteq R_i$ of attributes which satisfy the condition (1) is called superkey.

The attribute A in R_i is prime with respect to F, if A is a part of key of Sch_i, otherwise A is nonprime in R_i.

Let $Sch_i = (R_i, F)$ be a scheme where V and W are nonempty subsets of R_i, and A is an attribute in R_i. Attribute A is transitively dependent on V via W if all the following conditions are satisfied:

(1) $V \to W \in F^+$,
(2) $W \to V \notin F^+$(i.e. V is not functionally dependent on W),
(3) $W \to A \in F^+$, and
(4) $A \notin VW$ (where VW is the union of V and W).

A relation scheme $Sch_i = (R_i, F)$ is in third normal form [4] with respect to a set F of functional dependencies, if it is in first normal form and there is no nonprime attribute in R_i that is transitively dependent upon a key of Sch_i scheme. The database scheme Sch is in third normal form with respect to F

if every relation scheme Sch_i in Sch is in third normal form with respect to F.

A relation scheme $Sch_i = (R_i, F)$ is in Boyce-Codd normal form [5] with respect to a set F of functional dependencies, if it is in first normal form and for every nontrivial dependency $V \to A \in F^+$ the dependency $V \to R_i \in F^+$ takes place. That is, the left side of every nontrivial functional dependency is a superkey for Sch_i. A database scheme Sch is in Boyce-Codd normal form with respect to F if every relation scheme Sch_i in Sch is in Boyce-Codd normal form with respect to F.

Let $X \to Y \in F^+$ and $< X_0, X_1, ..., X_n >$ be the maximal derivation [6] of the set X under F. Let X_i be the first element which contains the set Y. Then the subsequence $< X_0, X_1, ..., X_i >$ is considered to be the *derivation* (not necessarily the maximal one) of the functional dependency $X \to Y$ under F.

The expression $X \to Y \in F^+$ takes place if and only if the derivation of $X \to Y$ under F exists.

3. Non-essential and recoverable attributes

In this section, we introduce the notion of two types of attributes: non-essential and recoverable, some of their properties are given which can be used to analyze database schemas. It is shown that the algorithms for determining the non-essential and recoverable attributes have a polynomial complexity.

Further, it is assumed that the set F of functional dependencies is reduced and divided into equivalence classes $F = F_1 \cup ... \cup F_n$.

It will be denoted by R_i the set of attributes of dependencies involved in class F_i, and by $PS(F_i)$ the set of left sides of dependencies F_i, and $F - F_i(C)$ will denote by the expression $(F - F_i) \cup \{X \to (Y - \{C\}) | X \to Y \in F_i\}$, for $i = \overline{1, n}$. Then, the non-essential attribute is defined as follows:

Definition 1. *Let F_i be an equivalence class, where $|F_i| > 1$ and $X \to YC \in F_i$. The attribute C is non-essential in the scheme $Sch_i = (R_i, F)$, if for every two left sides V and Z, where $V, Z \in PS(F_i)$, $V \to Z \in (F - F_i(C))^+$ takes place.*

It is not difficult to see that the non-essential attribute C is in the right side of only one dependency of equivalence class F_i. Otherwise, the set F would not be a reduced set of functional dependencies.

It should be mentioned that if C_1 and C_2 are non-essential attributes in the scheme Sch_i, then their union $C_1 \cup C_2$ is not necessarily non-essential.

Example 1. *Let $F = F_1 \cup F_2 \cup F_3$ be a given set of functional dependencies divided into three equivalence classes $F_1 = \{C_1 \to D\}$, $F_2 = \{C_2 \to D\}$ and*

$F_3 = \{AD \to B,\ AB \to C_1C_2\}$. It can be verified that both C_1 and C_2 are non-essential attributes in the scheme $Sch_3 = (\{A, B, C_1, C_2, D\},\ F)$, but their union $C_1 \cup C_2$ is not non-essential, because $AB \to AD \notin (F - F_3(C_1C_2))^+$.

Here and below, F is written in scheme Sch_i because the introduction of the set F_i would not be correct, due the set F_i^+ is a subset of a set of functional dependencies defined over the set $R_i$of attributes. In this context and considering the hypothesis of universal schema [7], the set $F$is presented in Sch_i, but it is meant the set of dependencies in F^+ satisfied by relations defined over the set R_i of attributes.

Definition 2. *The attribute A, where $A \in R_i$, is recoverable in $Sch_i = (R_i, F)$, if $(R_i - A) \to A \in (F - F_i)^+$ takes place.*

Unfortunately, likewise non-essential attributes, the union of recoverable attributes is not always recoverable.

From the definition of non-essential attribute, it appears a remarkable property of it: within the class of equivalence the free navigation of non-essential attributes on the right sides of the dependencies it is allowed. However, the closure of the set of dependencies remains intact. But, in the new set, the dependencies can become non-reduced.

Example 2. *Let $F = F_1 \cup F_2$ be a set of functional dependencies divided in two equivalence classes $F_1 = \{C \to B\}$ and $F_2 = \{AD \to B,\ AB \to DC\}$ of dependencies. The attribute C is non-essential in the scheme $Sch_2 = (\{A, B, C, D\},\ F)$, because $AB \leftrightarrow AD$ under $F - F_i(C)$. The set F is equivalent to G, where $G = G_1 \cup G_2$ and $G_1 = F_1$, but $G_2 = \{AD \to BC,\ AB \to D\}$. However, the dependency $AD \to BC$ is not reduced, because its substitution with $AD \to C$ does not affect the set G^+. It is easy to verify that the removed attribute B is recoverable in Sch_2.*

Therefore, the moving of non-essential attributes can be used to obtain an equivalent set of dependencies, but with fewer attributive symbols.

The algorithm for calculation of non-essential attributes is based on the following theorem:

Theorem 1. *Let $F = F_1 \cup ... \cup F_n$ be a minimum and reduced set of functional dependencies, and let $f : X \to YC \in F_i$ be a dependency. The attribute C is non-essential in Sch_i, if and only if for every left side $Z \in PS(F_i)$ the following $X \to Z \in (F - F_i(C))^+$ holds.*

Proof. (Necessity) The veracity of this statement follows directly from the definition of non-essential attribute. □

Proof. (Sufficiency) For the reverse statement is enough to show that for every left side $Z \in PS(F_i)$ the expression $Z \to X \in (F - f)^+$ takes place.

Indeed, the fact that $Z, X \in PS(F_i)$, shows that $Z \to X \in F^+$. But for the dependency f to be non-redundantly used in the derivation of dependence $Z \to X$ under $F - f$ it is necessary $Z \to X \in (F - f)^+$ to take place. □

The algorithm to calculate the non-essential attributes:

ATR_NEES(F,ClasEchivDep)
    $MatrAtrNees := 0$;
    for each $f : X \to Y \in F$ *do;*
        $i := ClasEchivDep(f)$;
        for each $C \in Y$ *do;*
            $m := 0$; $j := 0$;$X_j := X$;
            repeat
                $j := j + 1$;$X_j := X_{j-1}$;
                for each $V \to W \in (F - F_i(C))$ *do;*
                    if $V \subseteq X_j$ *then* $X_j := X_j \cup W$;
                    if $V \in PS(F_i)$ *then* $m := m + 1$;
                endfor;
            until $X_j = X_{j-1}$;
            if $|F_i| = m$ *then* $MatrAtrNees(i, C) := 1$;
        endfor;
    endfor;
end ATR_NEES.

In the described algorithm, $ClasEchivDep$ is an array showing the equivalence class of each dependency in F, and $MatrAtrNees$ is a two-dimensional array where for each class non-essential attributes are fixed. The closure of the set X of attributes under the set of dependencies $F - F_i(C)$ is calculated (*repeat* loop) in time $O(||F||)$ [2]. In the same loop it is determined whether the conditions of Theorem 1 are satisfied. Therefore, the *ATR_NEES* procedure takes $O(||F||^2)$ time.

It is not difficult to note that the algorithm for calculating the recoverable attributes has a temporal complexity similar to the procedure for determining non-essential attributes.

These constructions and statements will be used to determine the degree of normalization of the database schema. It is known that the problem of determining the degree of normalization is of exponential nature. First, the definitions of normal forms (second, third and Boyce-Codd normal forms) contain the key notion. But it is known that a relation can have an exponential

number of keys according to the number of attributes of the scheme. Second, the definitions of normal forms appeal to notions of prime and nonprime attributes, the concepts related again to the notion of key.

4. Sufficient conditions for normalization

In this section, it is shown that the sets of non-essential and recoverable attributes of a relational scheme are disjoint. In addition, it is demonstrated that any attribute that transitively depends on a key of scheme is recoverable in this scheme. New features in terms of non-essential and recoverable attributes for the schemes in third normal form are presented.

The main result of this section is to present and demonstrate sufficient conditions for a relational scheme to be in Boyce-Codd normal form. The proposed algorithm for testing whether a scheme is in Boyce-Codd normal form is of polynomial complexity.

Let $Sch_i = (R_i, F)$ be a relation scheme, and S and T are the sets of non-essential and recoverable attributes, respectively. Then the following assertion can be formulated:

Lemma 1. $S \bigcap T = \emptyset$.

Proof. It is supposed the opposite: $S \bigcap T \neq \emptyset$, that is, there is in the set R_i of attributes an attribute C, which is both non-essential and recoverable in the Sch_i. Let the attribute C be found in the right side of dependence $X \to YC$ of equivalence class F_i. Taking into account the non-essential attribute definition, the expression $X \to Z \in (F - F_i(C))^+$ holds for every left side Z of the set $PS(F_i)$ of left sides. Therefore, $X \to (R_i - C) \in (F - F_i(C))^+$. From the definition of recoverable attribute it follows that $(R_i - C) \to C \in (F - F_i)^+$ and from the fact that $(F - F_i) \subseteq (F - F_i(C))$, it occurs that $(R_i - C) \to C \in (F - F_i(C))^+$. Of $X \to (R_i - C) \in (F - F_i(C))^+$ and $(R_i - C) \to C \in (F - F_i(C))^+$, it follows that $X \to C \in (F - F_i(C))^+$. But this contradicts the assumption that the dependency $X \to YC$ is reduced. □

Lemma 2. *If an attribute A in R_i transitively depends on a key of the scheme $Sch_i = (R_i, F)$, then, A is recovered Sch_i.*

Proof. Because A transitively depends on a key of the scheme $Sch_i = (R_i, F)$, let it be X, then, there are a set $V \subseteq R_i$ of attributes, such that $X \to V \in F^+$, $V \to A \in F^+$, $V \to X \notin F^+$ and $A \notin XV$. Then there is the derivation $H =< V_0, ..., V_m >$ for dependency $V \to A$ under F. Since $V \to X \notin F^+$, then $X \subseteq V_m$ and the left side of each dependency used in the construction of H is not equivalent to X. Thus, $V \to A \in (F - F_i)^+$. But $V \subseteq (R_i - A)$ and hence $(R_i - A) \to A \in (F - F_i)^+$. □

Theorem 2. *Let $Sch_i = (R_i, F)$ be a relation scheme. Then*

a) if every nonprime attribute is non-essential, the scheme Sch_i is in third normal form.

b) if every prime attribute is recoverable, the scheme Sch_i is in third normal form.

Proof. Veracity of statements a) and b) arise from the definition of scheme in third normal form and lemmas 1 and 2. □

The statements given in Theorem 2 contain the concepts of prime and nonprime attributes. But, as mentioned in [1], the problem of determining whether the attributes are prime or not is an NP-complete one.

In terms of applicability, the assertion set out by the next theorem is more acceptable:

Theorem 3. *If every attribute A in R_i is not recoverable in scheme $Sch_i = (R_i, F)$, then $Sch_i = (R_i, F)$ is in Boyce-Codd normal form.*

Proof. Let the scheme $Sch_i = (R_i, F)$ not be in Boyce-Codd normal form. That is, there is at least one functional dependency $V \to A \in F^+$, where $VA \subseteq R_i$, $A \notin V$ and V is not a superkey for $Sch_i = (R_i, F)$. But in this case A transitively depends on a key of the scheme Sch_i and according to Lemma 2 A is recoverable. We have obtained a contradiction. □

Corollary 1. *If every attribute A in R_i that does not belong to any left side of dependencies in equivalence class F_i, is not recoverable in $Sch_i = (R_i, F)$, then the scheme $Sch_i = (R_i, F)$ is in third normal form.*

It should be mentioned that conditions of Theorem 3 and Corollary 1 guarantee the existence of schemes in Boyce-Codd normal form and third normal form, respectively. But not every scheme that is in Boyce-Codd normal form (third normal form) satisfies the conditions of Theorem 3 (Corollary 1).

For example, let $F = F_1 \cup F_2 \cup F_3 \cup F_4$ be a set o functional dependencies where $F_1 = \{KL \to B\}$, $F_2 = \{A \to K\}$, $F_3 = \{CE \to L\}$ and $F_4 = \{ADE \to B, AB \to CDE\}$. Although the attribute B is recoverable in $Sch_4 = (\{A, B, C, D, E\}, F)$, however the schema Sch_4 is in Boyce-Codd normal form.

Nevertheless the "fast" algorithms based on the Theorem 3 and Corollary 1 can be useful for database schemas analyzing.

Moreover, the corollary 1 permits application of a fairly simple algorithm for the synthesis of database schema. Indeed let it be given a minimum set of functional dependencies, divided into equivalence classes $F = F_1 \cup ... \cup F_n$. It is know, that such a set can be achieved by $O(||F||^2)$ operations [9].

Step 1. The set Z_i of all attributes is built, which are located in the right and not in the left sides of dependencies in the current equivalence class F_i.

Step 2. From the set Z_i the next attribute A is selected.
Step 3. It is checked if X $(R_i - A) \to A \in (F - F_i)^+$ takes place. If so, then the $R_i := R_i - \{A\}$ and $Z_i := Z_i - \{A\}$ are set. Steps 2-3 are executed until all attributes in Z_i are examined.
Step 4. The right sides of the dependencies of equivalence class F_i are substituted with R_i. Steps 1-4 are executed for all equivalence classes of functional dependencies.
Step 5. The set of functional dependencies is reduced.

Each equivalence class will represent a relational scheme and the left sides of dependencies in the class will be the possible keys.

Since all recoverable attributes in the right sides of the dependencies are removed, the database schema obtained by the application of described steps will consist of relational schemes in third normal form.

It is not difficult to show that the complexity of this algorithm is the same as the complexity of the algorithm proposed by Bernstein [3] - $O(||F||^2)$.

5. Conclusions

It should be noted that the problem of relational schemes analysis is the most poorly studied. It is more difficult to identify the properties of existing databases rather than building a database with such features.

Researches are needed to determine various classes of attributes that directly affect the quality properties of the database. Obviously the question arises, if in order to recognize a normal form of a scheme is essential the determination of the prime and nonprime attributes or the determination of all keys.

It must be also investigated whether the identification of prime attributes implies searching all the keys. It is obvious, that the acceptable case, in terms of complexity of the problem is the non-affirmative one. Thus, the investigations are required to determine the equivalent characteristics of these entities, but that could be calculated in polynomial time.

Another aspect of the analysis problem lies in determining the normalization degree of a scheme. The data presented in the literature that touch this aspect is not studied enough that can be used for database testing. Investigations are needed to improve algorithms and development of new classes of models that would present features and entities at least sufficient, if not necessary and sufficient for normal forms.

References

[1] Beeri C., Bernstein Philip A., *Computational Problems Related to the Design of Normal Form Relational Database.* ACM Trans. Database Syst., V.301, N. 4, pp.752-766, 1983.

[2] Beeri C., Bernstein Philip A., *Computational Problems Related to the design of Normal Form Relations Schemes.* ACM Trans. Database Syst., V.4, N 1, p.30-59, 1979.

[3] Bernstein Philip A., *Synthesizing Third Normal Form Relations from Functional Dependencies.* ACM Trans. Database Syst., V.1, N. 4, p.277-298, 1976.

[4] Codd E.F., *Further Normalization of the data base relational model.* Data Base Systems, R. Rustin (ed), Prentice Hall, p. 33-64, 1972.

[5] Codd E.F., *Recent Investigation in Relation Data Base Systems.* IFIP Congress, p.1017-1021, 1974.

[6] Cotelea Vitalie, *An inference model for functional dependencies in database schemas.* Meridian Ingineresc, N.3, p. 89-92, 2009.

[7] Kent W., *The Universal Relation Revised.* ACM Trans. Database Syst., V.8, N 4, p.644-648, 1983.

[8] Lucchesi C.L., Osborn S.L., *Candidate Keys for Relations.* Jour. of Comput. and Syst. Sci., N.17, p.270-279, 1978.

[9] Maier D., *The Theory of Relational Database.* Computer Science Press, 637 p., 1983.

[10] Maier D., *Minimum Cover in the Relational Database Model.* Jour. of ACM, V.27, N. 4, p 664-674, 1980.

[11] Osborn Sylvia L., *Testing for Existance of a Covering Boyce-Codd Normal Form.* Information Processing Letters, V.8, N.1, p. 11-14, 1979.

[12] Paredaens J., *About Functional Dependencies in a Database Structure and their Coverings.* PhillipsMBLE Lab. Report 342, 1977.

[13] Worland Peter B., *An Efficient Algorithm for 3NF Determination.* Information Science, V.167, N.1-4, p.177-192, 2004.

Faculty of Cybernetics, Statistics and Economic Informatics, Academy of Economic Studies of Moldova, 59 Banulescu-Bodoni street, Chisinau, Republic of Moldova

E-mail address: vitalie.cotelea@gmail.com

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVII**, Number 1, 2012

DESIGNING AND IMPLEMENTING APPLICATIONS FOR HEARING-IMPAIRED CHILDREN

CĂTĂLINA HARASIM

Abstract. Hearing-impaired children need extra help when learning new words and concepts. Because they cannot truly understand and perceive all the information they hear or read, a very effective method is to use images and sign language during the learning process. *aSIGNment* helps in providing an education for hearing-impaired children, starting from preschool, by using technology to bring a new way in meeting their needs during the learning process and in obtaining the supervision of a qualified educator.

1. Introduction

We all learn throughout our lives. Human beings have five senses as their main assets in experimenting the world around them, learning and communicating. A child is open-minded and eager to learn as much as possible. His understanding and integration into society are much better if he receives as much information as possible.

Hearing is crucial in speech and language development. It is extremely important to diagnose and treat hearing loss at an early age. A hearing-impaired child can normally develop speech and language if he benefits of a hearing aid device before the age of 6 months [1].

This is ideal, but not possible in many cases. Therefore, one of the first problems that arise is schooling. Hearing-impaired children need extra help when learning new words and concepts. Physical objects normally do not pose problems, but abstract concepts such as time, feelings and thoughts are harder to explain [3]. The solution proposed, *aSIGNment*, is a software that works towards providing education to every child with hearing impairment around the world (starting from preschool, ages 3-6), by optimizing education systems

Received by the editors: January 14, 2012.

2010 *Mathematics Subject Classification.* 68N99, 68U99.

1998 *CR Categories and Descriptors.* K.3.1 [**Computing Milieux**]: Computer Uses in Education – *Computer-assisted instruction*; J.4 [**Computer Applications**]: SOCIAL AND BEHAVIORAL SCIENCES – *Psychology*; J.3 [**Computer Applications**]: LIFE AND MEDICAL SCIENCES – *Health*.

Key words and phrases. hearing impairment, sign language, preschool children, education.

already in place. Similar solutions (e.g. ABC Deaf Software, IDRT software) are mostly targeted on hearing development and are used to sustain teaching or as self-study. Compared to them, *aSIGNment* uses technology, especially Microsoft Silverlight and Microsoft .NET, to present the information in a way in which hearing-impaired children can understand in better — images and videos of sign language. Children will learn in a specialized organization or at home, but always under the supervision of a qualified educator.

2. Problem Definition

Communication is a necessary skill for us to function well throughout life. We tend to take it for granted that children will develop their language and communication skills naturally and in a predictable way. Being deaf can make that process more challenging, but with the right support, commitment and encouragement from both families and professionals, deaf children can learn to communicate as effectively as other children [8].

It is important to make a clear distinction between deafness and mental deficiencies [1]. Hearing-impaired children bear the burden of their handicap and may become shy, anxious, afraid of people around them, but this is definitely not enough to assume the existence of retardation.

Several relevant facts about deafness are presented below:

- In 2005, about 278 million people had moderate to profound hearing impairment. 80% of them live in low and middle-income countries [10].
- An estimated 12.5% of children and adolescents aged 6–19 years of the United States have permanent hearing damage [2].
- About 2 to 3 out of every 1000 children in the United States are born deaf or hard-of-hearing. Nine out of every 10 children who are born deaf are born to parents who can hear [9].
- Only 1 out of 5 people who could benefit from a hearing aid actually wears one [9].

A key aspect here is that most hearing impairment is not genetically inherited, but rather acquired as the effect of disease or trauma. This means that families of deaf children usually have no experience with the challenges that will follow.

In order to better understand the learning process of a hearing-impaired child and the methods that are used, we have had several discussions with Mrs. Cristina Dohotaru, speech psychotherapist for preschool children at Şcoala Specială pentru Surzi Cluj-Napoca. She explained that many families choose to enroll their hearing-impaired child at mainstream schools, mainly because they do not have enough knowledge about this problem, but also due to financial and geographical reasons — specialized institutions are not available

nearby. Even in special schools, most of the times the emphasis falls on improving children's hearing, despite the fact that they cannot truly understand and perceive all the information they hear or read. For many deaf children, sign language is the only way of efficient communication and learning. It is the critical first step to communication and to the later on development of literacy and spoken language skills [4]. Sign language also provides a way of interacting with other children, which will put the basis for social integration and a healthy self-image.

3. *aSIGNment* — Educational Software for Hearing-Impaired Children

Our solution intends to help hearing-impaired children (starting from preschool) to adapt more quickly to life, by learning them the means to make themselves understood by the others — the sign language. *aSIGNment* is an educational application which can be successfully used in any kindergarten for hearing-impaired children, but also by any other such child from home, as it can be seen in Figure 1 [3].

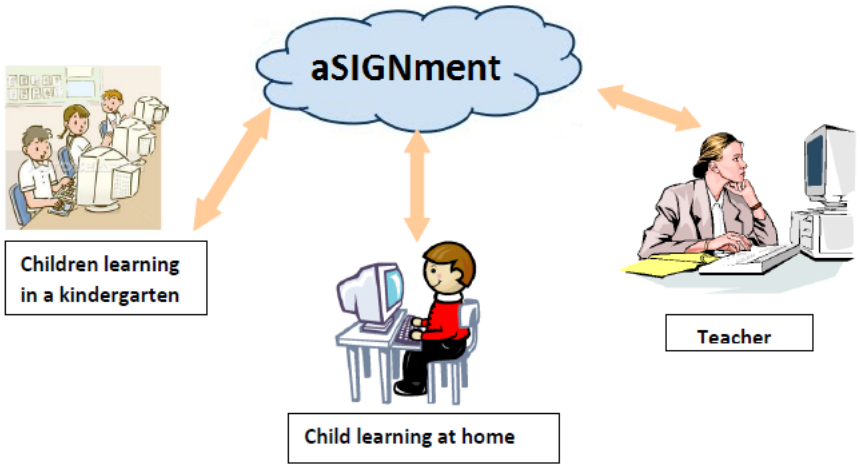


Figure 1. Usage of *aSIGNment*

aSIGNment promotes learning by playing, is accessible from any Internet connected computer, and offers real-time communication. Our approach has several important advantages:

- Permanent monitoring and development under the supervision of a qualified educator.

- Child evolution can be enhanced through specially designed lessons and activities: the teacher can create new lessons and games, can choose which activities to assign to a child based on his evaluation.
- A child who cannot enroll in a special kindergarten or school (due to financial or geographical reasons) has access to education corresponding to his special needs.
- The multiple mice experience in lessons and games offers real communication between children and between child and educator.

The project consists of two applications: one for the children (*aSIGNment Learning*) and one for the teacher (*aSIGNment Teaching*).

The lessons and games were designed starting from Mrs. Dohotaru's advices and explanations regarding the teaching methods used for young hearing-impaired children.

3.1. **Special Features and Technologies.** Since the emphasis falls on learning by playing, the information is presented through a simple, friendly and attractive interface, such that it is appealing to children. A demo in a kindergarten has produced positive feedback, from both children and Mrs. Dohotaru.

Using *aSIGNment Teaching*, the teacher can create new lessons or just modify an existing one, according to the needs of the children, by using a simple menu with all the needed options. There are several patterns of lessons, organized into topics (such as animals or vegetables), according to the various teaching methods that a teacher uses in class. Each pattern can contain text, images and videos of sign language. For example, a pattern requires the children to choose between several images in order to correctly associate a video with a word expressed in sign language with the image that represents that word.

The lessons the teacher creates are presented to the children in the form of educational games, a very effective method for them to learn at this age. In order to create lessons in a more interactive way, the teacher can upload his own photos and videos and then use them.

The children can participate to the lessons in the classroom or from their home computer. The teacher will still supervise their work, analyze their progress by viewing statistics and create corresponding lessons.

A special category of lessons and games contains multiple mice interaction. Children develop social skills and team spirit by working on the same lesson or game together with other children, each having his own mouse device.

Some of these functionalities are illustrated in Figures 2, 3 and 4.

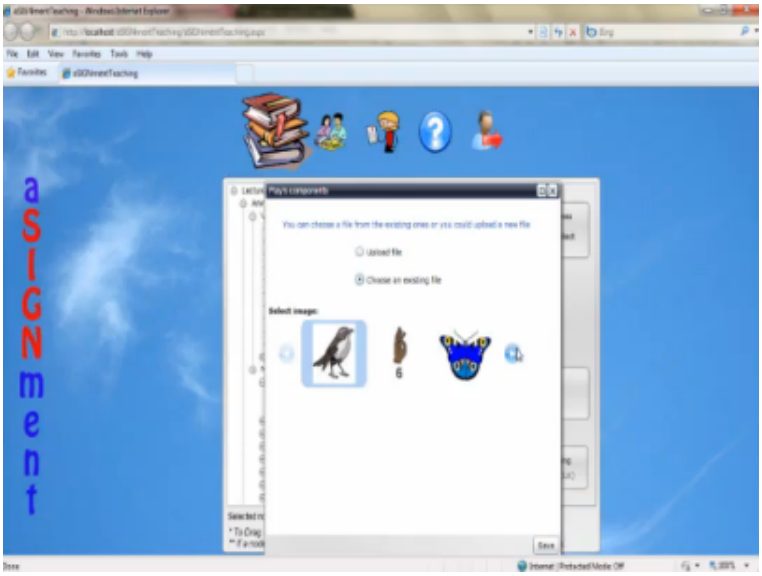


FIGURE 2. *aSIGNment Teaching* — Choosing content for lessons

FIGURE 3. *aSIGNment Teaching* — Viewing results and analyzing progress

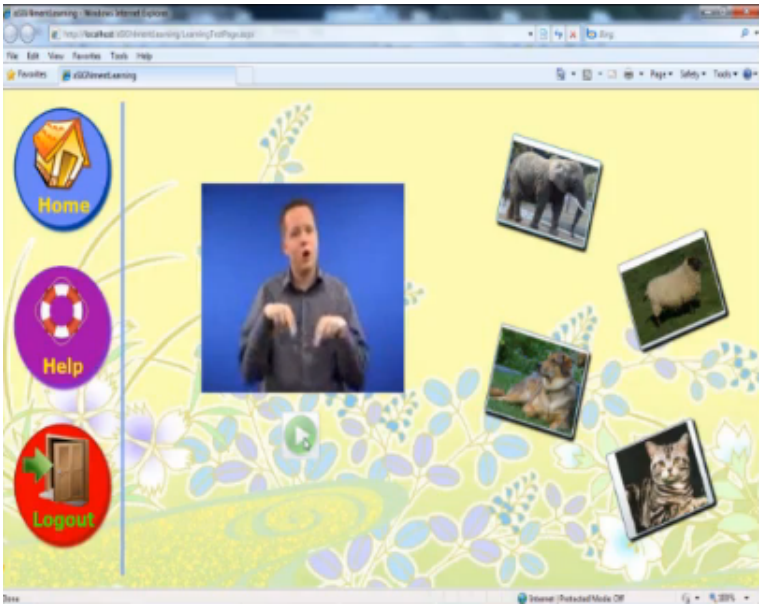


FIGURE 4. *aSIGNment Learning* — Lesson example

3.2. **Technical Aspects.** Both *aSIGNment* applications are based on a 3-Layer architecture, together with the database connection. Each layer communicates only with its neighbors: serves as a client for the layer below and as a services provider for the layer above. Each layer is briefly described below:

- **aSIGNmentLearning** and **aSIGNmentTeaching** are Presentation Layers. They contain the user oriented functionality responsible for managing user interaction with the system and components that provide a common bridge into the core business logic encapsulated in the Business Layer [7].
- **aSIGNmentWFCService** is the Business Layer, which implements the core functionality of the system and encapsulates the relevant business logic. It also exposes service interfaces that the Presentation Layers can use [7].
- **aSIGNmentRepository** is the Data Access Layer and provides access to data. This layer exposes generic interfaces that the components in the Business Layer can consume [7].

The architecture of the solution is shown in Figure 5.

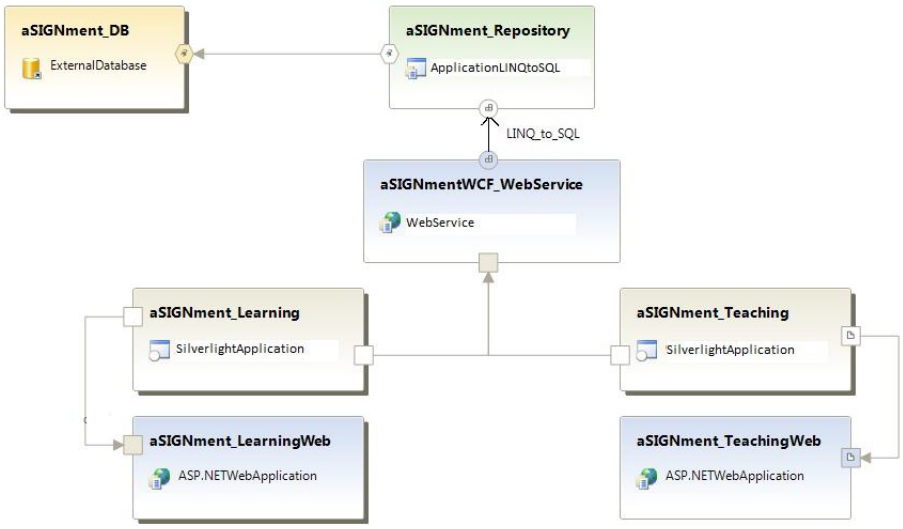


FIGURE 5. *aSIGNment* Architecture

aSIGNment was developed using Visual Studio 2010 and the main technologies used were .NET 4.0, Silverlight 4, Windows Communication Foundation (WCF) Web Service, Windows MultiPoint Mouse SDK 1.5, LINQ to SQL and SQL Server for the database connectivity.

aSIGNmentLearning and *aSIGNmentTeaching* are Silverlight projects that expose information to the end user. Silverlight helps in creating interactive applications, especially important when considering their main users (children are attracted by nice designs and easily learn by "playing"). Silverlight enables Web-based applications to deliver the business functionality users demand with a modern, efficient user interface. This layer communicates with the Business Layer through a Web service.

aSIGNmentWCFService communicates with the Data Access Layer directly through method calls. WCF itself is designed in accordance with service oriented architecture principles to support distributed computing where services are consumed by clients. In this case, the consumer is a Silverlight project (the Presentation Layer). The Presentation Layer uses a WCF service reference to allow the Silverlight project to communicate by asynchronous calls with the Data Access Layer, according to its operation contracts and data contracts. The data contracts expose the structure of the entities from the data layer part. This makes it simple for the Silverlight application's controls to be bound to the instances of the entities and their properties. The operation

contracts define the methods that the Silverlight application can invoke on the WCF service [6].

aSIGNmentRepository accesses the underlying data store, an SQL Server database. It uses LINQ to SQL to connect to the database, which brings several important advantages. LINQ to SQL provides a run-time infrastructure for managing relational data as objects without losing the ability to query. It does this by translating language-integrated queries into SQL for execution by the database and then translating the tabular results back into the defined objects. The application is then free to manipulate the objects while LINQ to SQL stays in the background tracking the changes automatically and taking care of data consistency [5]. There is no need to write code to define the entities in the application, LINQ to SQL uses specific attributes that map the database tables into the classes it generates. Further on, the WCF Web service reference makes them known to the rest of the application. In this way, the layers share a model, but not as actual code written by a programmer, but by some sort of common "language" that is taken care of by the technologies used.

4. Conclusions and Future Work

Our project, *aSIGNment*, intends to provide an education for hearing-impaired children, according to their needs, and it is still at its basic level of functionalities. It can further be improved, for example, by adding new lesson patterns or by allowing the teacher to create his own patterns.

An extension of *aSIGNment* is currently under development, in collaboration with an educational NGO, Media Kinder. This extension is an online general knowledge contest for hearing-impaired children. The questions are formulated by specialized teachers, according to the information they learn in school, and the children compete in teams of two or three. There is a separate set of questions for each grade, from the 2^{nd} to the 6^{th} grade.

Our final aim is to obtain a software product which is used by schools and institutions for deaf children, that will truly help with their education and integration.

5. Acknowledgments

This research was supported by Babeş Bolyai-University performance grant, no. 30005/10.01.2011.

I want to thank Roxana Chelemen, Oana Iova, Tania Nemeş and Assoc. Prof. Dr. Simona Motogna for all their collaboration and support throughout this project.

References

1. M. D. Avramescu, *Defectologie şi logopedie — Ediţia a 3-a*, Editura Fundaţiei România de Mâine, Bucureşti, 2007.
2. Centers for Disease Control and Prevention, `www.cdc.gov`.
3. R. Chelemen, C. Harasim, O. Iova, T Nemeş, and S. Motogna, *aSIGNment*, Proceedings of the National Symposium Zilele Academice Clujene (2010), 182–187.
4. T. V. Malloy, *Sign language use for deaf, hard of hearing, and hearing babies: the evidence supports it*, American Society for Deaf Children, 2003.
5. Microsoft, *LINQ to SQL: .NET language-integrated query for relational data*, `www.msdn.microsoft.com/en-us/library/bb425822.aspx`.
6. ———, *What is Windows Communication Foundation*, `www.msdn.microsoft.com/en-us/library/ms731082.aspx`.
7. ———, *Microsoft Application Architecture Guide, 2^{nd} edition*, 2009.
8. National Deaf Children's Society, *Communicating with your deaf child*, 2010.
9. National Institute of Deafness and Other Communication Disorders, `www.nidcd.nih.gov`.
10. World Health Organization, `www.who.int`.

Department of Computer Science, Faculty of Mathematics and Computer Science, Babeş-Bolyai University, Kogălniceanu 1, Cluj-Napoca 400084, Romania
E-mail address: `catalina.harasim@gmail.com`

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVII**, Number 1, 2012

PRINCIPLES OF ACTION SEMANTICS FOR FUNCTIONAL PROGRAMMING LANGUAGES

VILIAM SLODIČÁK, VALERIE NOVITZKÁ

Abstract. In this paper we present a short introduction into foundations of action semantics and its application in functional paradigm. We discuss the use of action semantics for functional paradigm. The obtained results are demonstrated on the particular well-known example from informatics - the Fibonacci numbers. The computation of Fibonacci numbers has been implemented in the object-oriented functional language OCaml and the description of the program is given in action semantics.

1. Introduction

Action semantics is one of the newest methods for defining the meaning of constructions of the programming languages. Formal definition of programming language is an inseparable part of the definition of every programming language. It allows us to understand the behavior of programs by unambiguous way. Traditional methods - an operational and denotational semantics belong in the group of the most used semantical methods. The operational semantics (a semantics of small steps) plays a crucial *rôle* in implementation of the programming language, because it concerns the details of the program execution. Denotational semantics, often called also mathematical semantics, expresses the meaning of programs by functions over the mathematical structures (domains). It has been used in the design of programming languages, but its mathematical complexity is quite hard for IT experts to understand. The action semantics avoids the disadvantages of the former methods. It arises from the denotational semantics but it provides the meaning of programs by user-friendly style. It uses English phrases so the definitions of semantics of

Received by the editors: February 4, 2012.

2010 *Mathematics Subject Classification.* 68Q55.

1998 *CR Categories and Descriptors.* F.3.2 [**Logics and meanings of programs**]: Semantics of Programming Languages – *Partial evaluation*; D.3.3 [**Programming languages**]: Language Constructs and Features – *Recursion.*

Key words and phrases. Action semantics, functional paradigm, actions, semantical description.

the program constructions are more readable. Action semantics is fully equivalent with other semantical methods, such as denotational semantics, natural semantics, operational semantics or axiomatic semantics [8].
Until now the action semantics has been defined for the imperative languages. But its principles are also appropriate for defining the semantics of the languages of functional paradigm. In our article, we present the principles of action semantics for the new functional object-oriented language - *OCaml* [1, 4]. The results obtained are demonstrated on the example of computation the Fibonacci numbers where we apply the results from the previous works [13, 14] about the recursion. In the section 2 we describe fundamentals of action semantics. In the next section we formulate some basic principles of action semantics in functional paradigm. The last section presents an example of applying the action semantics in functional paradigm.

2. Basic notions about Action semantics

The framework of action semantics [7] was initially developed at the University of Aarhus by Peter D. Mosses, in collaboration with David Watt from University of Glasgow. One of its main advantages over other frameworks is pragmatic: action-semantic descriptions can scale up easy to real programming languages [10, 16]. Action semantics deals with the three kinds of semantic entities: actions, yielders and data. Fundamentals of action semantics are actions which are essentially dynamic computational entities. They represent the computational behavior by using the values passed to them to generate new values that reflect changes in the state of the computation. In other words - the performance of an action directly represents the information of processing the behavior and reflects the gradual step-wise nature of computation: each step of an action performance may access and/or change the current information.

Other semantic entities used in action semantics are yielders and data. The information processed by actions consists of items of data, organized in structures that give access to the individual items. Data could contain:

- *mathematical entities* (e.g. truth values, numbers, characters, strings, lists, sets, and maps);
- *computational entities* (tokens and cells);
- *compound entities* (messages and contracts).

Yielders are another entities; they can be evaluated to yield data during action performance. The data yielded may depend on the current information:

- the given transients;
- the received bindings;

- the current state of the storage (not applied in functional programming).

The actions are main kind of entities; the yielders and data are subsidiary. The notation used for specifying actions and the subsidiary semantic entities is called action notation [7]. In action semantics, the semantics of a programming language is defined by decomposition of program phrases to actions. The performance of these actions relates closely to the execution of the program phrases. Primitive actions can store data in storage cells, bind identifiers to data, compute values, test truth values, etc. [11].

A performance of an action which may be a part of an enclosing action either:

- *completes*, corresponding to normal termination;
- *escapes*, corresponding to exceptional termination;
- *fails*, corresponding to abandoning an alternative;
- *diverges*, corresponding to deadlock.

2.1. **Action semantics facets.** The different kinds of information give rise to the so called *facets* of actions which have been classified according to [7]. They are focusing on the processing of at most one kind of information at a time:

- *the basic facet*, processing independently of information (control flows);
- *the functional facet*, processing transient information (actions are given and give data);
- *the declarative facet*, processing scoped information (actions receive and produce bindings);
- *the imperative facet*, processing stable information (actions reserve and dispose cells of storage, and change the data stored in cells);
- *the communicative facet*, processing permanent information (actions send messages, receive messages in buffers, and offer contracts to agents) [7].

2.2. **Action notation and combinators.** The standard notation for specifying actions consists of primitive actions and action combinators. Examples of action combinators are depicted on the following figures: action combinator `and` (Fig. 1), action combinator `and then` (Fig. 2), action combinator `then` (Fig. 3) and action combinator `or` (Fig. 4). In diagrams, the scoped information flows from left to the right whereas transients flow from top to the bottom.

- A_1 `and` A_2 (Fig. 1) allows the performance of the two actions to be interleaved. No control dependency in diagram, so actions can be performed collaterally.

- A_1 `and then` A_2 (Fig. 2) performs the first action and then performs the second one.
- A_1 `then` A_2 (Fig. 3) performs the first action using the transients given to the combined action and then performs the second action using the transients given by the first action. The transients given by the combined action are the transients given by the second action.
- A_1 `or` A_2 (Fig. 4) arbitrarily chooses one of the subactions and performs it with given transients and received bindings. If the chosen action fails, it perform the other subaction with original transients and bindings.

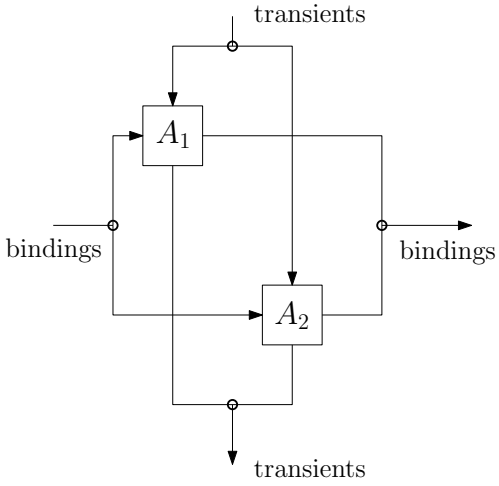


FIGURE 1. Action combinator A_1 *and* A_2

The data entities consist of mathematical values, such as integers, Boolean values, and abstract cells representing memory locations, that embody particles of information. Sorts of data used by action semantics are defined by algebraic specifications. Yielders encompass unevaluated pieces of data whose values depend on the current information incorporating the state of the computation. Yielders occur in actions and may access, but they can not change the current information.

The standard notation for specifying actions consists of primitive actions and action combinators. Action combinators combine existing actions, normally using infix notation, to control the order which subactions are performed in as well as the data flow to and from their subactions. Action combinators are used to define sequential, selective, iterative, and block structuring control flow as well as to manage the flow of information between actions. The

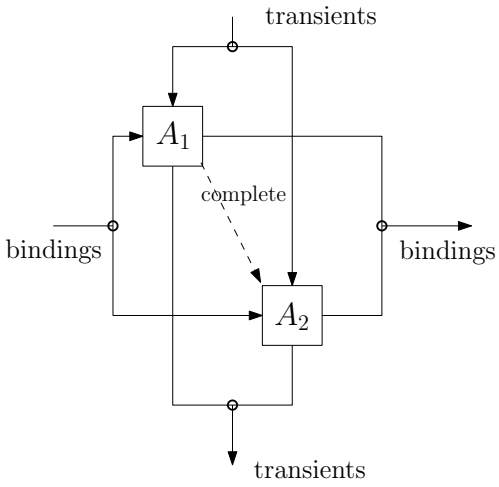


FIGURE 2. Action combinator A_1 *and then* A_2

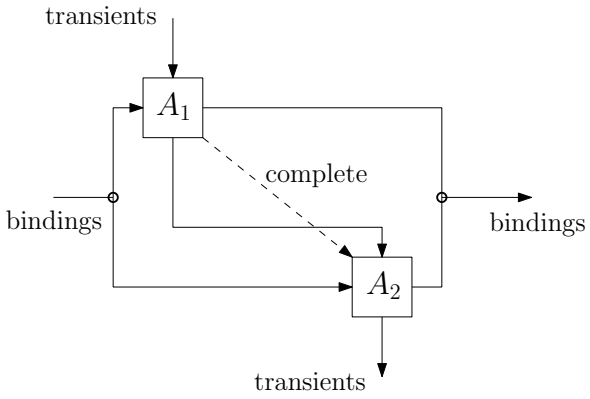


FIGURE 3. Action combinator A_1 *then* A_2

standard symbols used in action notation are ordinary English words. In fact, action notation is very near to natural language:

- terms standing for actions form imperative verb phrases involving conjunctions and adverbs, e.g. `check it and then escape`;
- terms standing for data and yielders form noun phrases, e.g. `the items of the given list`.

These simple principles for choice of symbols provide a surprisingly grammatical fragment of English, allowing specifications of actions to be made fluently readable. The informal appearance and suggestive words of action notation

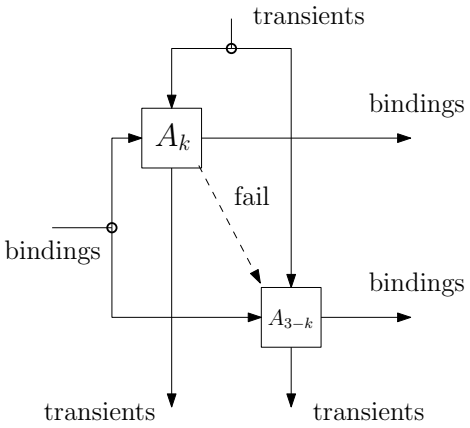


FIGURE 4. Action combinator A_1 *or* A_2 (for $k = 1$ or $k = 2$)

should encourage programmers to read it. Compared to other formalisms, such as λ-notation, action notation may appear to lack conciseness: each symbol generally consists of several letters, rather than a single sign. But the comparison should also take into consideration that each action combinator usually corresponds to a complex pattern of applications and abstractions in λ-notation. In any case, the increased length of each symbol seems to be far outweighed by its increased perspicuity.

3. Action semantics in functional paradigm

Action semantics can be successfully used also for the description of functional programs [15]. In action semantics we use generally three main actions for the description of programming languages:

- **execute**$[\![$*statement*$]\!]$ - used for executing of statements;
- **elaborate**$[\![$*declaration*$]\!]$ - used with declarations;
- **evaluate**$[\![$*expression*$]\!]$ - used for evaluating expressions.

In functional paradigm we use only two main actions: *evaluate* and *elaborate*. Action *execute* is not used in functional paradigm; it is a standard action for imperative paradigm. Typical for functional programs is that they do not deal the storage. Therefore we will not use actions of imperative facet for allocating memory locations, storing values and getting values from cells in memory in our action semantics descriptions of functional programs.
The actions **evaluate** and **elaborate** we usually use in the form

$$\textbf{evaluate}[\![e]\!]s\,[n \mapsto val_0]\,,$$

where e is an expression to be evaluated in the input state denoted s where n has the value val_0.
Important for functional paradigm is the evaluation of the expressions and the elaboration of the functions. To allow referring them in the program code, they are associated to names (identifiers). These associations are called bindings. A binding can be *global*, when declared at the top level of the source code, or *local*, when declared in a *let* or *letrec* expressions that contain it. The difference between *let* and *letrec* expressions is that in the latter one a mutual recursion is allowed. We provide this description of evaluation of simple expression:

elaborate⟦`let` I`:Var = E:Expression`⟧ =
  evaluate ⟦ `E` ⟧
    `then bind` I `to the given value`

After declaration we are able to use it anytime in our program. The value is bound to its identifier, so we can get the value of this expression simply by using *evaluate* action:

evaluate⟦ I`:Var` ⟧ =
  `give the value bound to` I

Description of a function with one argument should seem like this:

elaborate⟦`let` I_f`:Var` I_{p1}`:Var = E:Expression`⟧ =
  evaluate⟦`E`⟧
    `then bind` I_f `to the given value`

In the expression E the parameter of the function is used. The value of the function we can get simply by action *evaluate*:

evaluate⟦ I_{p1}`:Var` ⟧ =
  `give the value bound to` I_{p1}

General definition for a function with two or more arguments:
elaborate⟦`let` I_f`:Var` $< I_p$`:Var` $>^+$ `= E:Expression`⟧ =
  evaluate⟦`E`⟧
    `then`
  `bind` I_f `to the given value`

3.1. **Related work.** In 1997, S. B. Lassen started to develop the functional part of a theory of action semantics for reasoning about programs. Action

notation, the specification language of action semantics, was given an evaluation semantics, and operational techniques from process theory and functional programming have been applied in the development of a versatile action theory [3]. Peter D. Mosses showed in [6] the functional action notation which has extended the basic action notation and data notation with a few primitive actions, one new combinator and some notation for yielders. This notation belongs to the group of languages based on composition of functions, without exlicit mention of arguments, such as FP. But there is also another group of functional languages based on application of functions to arguments, such as Standard ML. On the other hand, David A. Watt presented in [17] an action-semantics description of Standard ML, as evidence for the claimed merits of action semantics.

4. Example

We present the description of functional program in action semantics at the well-known algorithm: Fibonacci numbers. In mathematics, the Fibonacci numbers are the numbers in the integer sequence where the first two Fibonacci numbers are 0 and 1 (sometimes first two Fibonacci numbers are considered 1 and 1), and each subsequent number is the sum of the previous two [2, 5, 12]. The construction of the sequence of Fibonacci numbers is as follows:

- $F_0 = 0$, sequence is (0);
- $F_1 = 1$, sequence is (0, 1);
- $F_2 = F_0 + F_1 = 0 + 1 = 1$, sequence is (0, 1, 1);
- $F_3 = F_1 + F_2 = 1 + 1 = 2$, sequence is (0, 1, 1, 2);
- $F_4 = F_2 + F_3 = 1 + 2 = 3$, sequence is (0, 1, 1, 2, 3);
- $F_5 = F_3 + F_4 = 2 + 3 = 5$, sequence is (0, 1, 1, 2, 3, 5);
- *etc.* ...

In mathematical terms we define the Fibonacci sequence by the linear recursive function:

$$F_n = \begin{cases} 1 & \text{if } n = 0; \\ 1 & \text{or } n = 1; \\ F_{n-1} + F_{n-2} & \text{otherwise } (n > 1). \end{cases}$$

The recursive function for the calculation of Fibonacci numbers is based on the "Divide et Impera" method. In the language *OCaml* it has the form:

```
let rec fibDnC(n) =
  if (n==0 || n==1)
       then 1
  else
       fibDnC(n-1) + fibDnC(n-2);;
```

4.1. **Description in action semantics.** We use a substitution for the term that calculates the Fibonacci number.

Let expression E be:

$$E = \texttt{if } (n == 0 \;||\; n == 1) \texttt{ then } 1 \texttt{ else } fibDnC(n-1) + fibDnC(n-2).$$

Next we elaborate the function $fibDnC(n)$

elaborate $[\![$ `let rec` *fibDnC (n)* $= E]\!] =$
`recursively bind` *fibDnC* `to`
`closure of`
`abstraction of`
evaluate$[\![E]\!] =$
`recursively bind` *fibDnC* `to`
`closure of`
`abstraction of`
evaluate$[\![n]\!]$
`and then`
`give the` $TruthValue$ `of`
(`the given number is equal to the number` 0
`or`
`the given number is equal to the number` 1)
`then`
`check the given` $TruthValue$
`and then`
`give the number` 1
`or`
`check not the given` $TruthValue$
`and then`
`give the sum of`
(**elaborate**$[\![fibDnC(n-1)]\!]$
`and`
elaborate$[\![fibDnC(n-2)]\!]$)

The value of argument given by expression n in function $fibDnC(n)$ we get in the action **evaluate**:

evaluate $[\![n]\!] =$
`give the value bound to` n

where the value of the input expression is simply evaluated and returned as output value.

4.2. **The description of an example in Action semantics.** We show a partial elaboration of the function $fibDnC(n)$ in the state for the input argument $n = 5$.

elaborate $[\![$*fibDnC (n)* $= E]\!]s\,[n \mapsto 5] =$
 give the value bound to
  closure of
   abstraction of
    evaluate$[\![E]\!]s\,[n \mapsto 5] =$

 give the value bound to
  closure of
   abstraction of
    give the number 5
      and then
    give the $TruthValue$ of
    (the given number is equal
       to the number 0
      or
     the given number is equal
       to the number 1)
    then
     check not the given $TruthValue$
        and then
         give the sum of
          (**elaborate**$[\![fibDnC(n)]\!]s\,[n \mapsto 4]$
            and
           elaborate$[\![fibDnC(n)]\!]s\,[n \mapsto 3]$)

Next step is the elaboration of the function for the input values $n = 4$ and $n = 3$.

The elaboration of the function $fibDnC(n)$ is given by recursive evaluation of the Fibonacci numbers calculation. Here we omit some steps and show only the final step after evaluation of the given terms. Here, in the final step of description the function $fibDnC(n)$ is being called with the input value $n = 0$.

elaborate $[\![$*fibDnC (n)* $= E]\!]s\,[n \mapsto 0] =$
 give the value bound to
  closure of
   abstraction of
    evaluate$[\![E]\!]s\,[n \mapsto 0] =$

```
give the value bound to
  closure of
    abstraction of
     give the number 0
       and then
     give the TruthValue of
     (the given number is equal
        to the number 0
       or
      the given number is equal
        to the number 1)
     then check the given TruthValue
         and then give the number 1
```

In the following description we replaced the clause `the given number` with the concrete values by reason of showing partial results in the evaluation.

```
give the value bound to
  closure of
    abstraction of
          give the sum of
           (the number 1 and the number 1)
       and then
          give the sum of
           (the number 2 and the number 1)
       and then
          give the sum of
           (the number 3 and the number 2)
       and then
          give the sum of
           (the number 5 and the number 3) =
give the value bound to
  closure of
    abstraction of
        give the number 8
```

The result of the function $fibDnC(n)$ for the input value $n = 5$ is equal to 8.

5. Conclusion

In this article we have formulated new application area of action semantics for functional programming languages. We presented our approach on the Fibonacci numbers computation function which is traditionally used for illustration of the recursion in functional programming [9].

In the future we extend our approach to the object-oriented features of functional programming language *OCaml* and to construct a categorical form of action semantics which can be useful to observe the behavior of functional programs in coalgebraic terms.

Acknowledgement

This work has been supported by the Slovak Research and Development Agency under the contract No. APVV-0008-10: Modelling, simulation and implementation of GPGPU-enabled architectures of high-throughput network security tools.

References

[1] Hickey, J. *Introduction to Objective Caml*. Cambridge University Press, 2008.
[2] Koubková, A., and Pavelka, J. *Introduction to theoretical informatics*. MatFyzPress, Charles University, Prague, 2005. (in Czech).
[3] Lassen, S. Action semantics reasoning about functional programs. *Journal Mathematical Structures in Computer Science Vol. 7*, Issue 5 (October 1997).
[4] Leroy, X. The objective caml system release 3.12. documentation and user's manual. Tech. rep., Institut National de Recherche en Informatique et en Automatique, 2008.
[5] Matoušek, J., and Nešetřil, J. *Kapitoly z diskrétní matematiky*. Nakladatelství Karolinum, Praha, Univerzita Karlova v Praze, 2000.
[6] Mosses, P. *Action Semantics*. Cambridge University Press, 2005.
[7] Mosses, P. D. Theory and practice of action semantics. In *In MFCS '96, Proc. 21st Int. Symp. on Mathematical Foundations of Computer Science* (1996), Springer-Verlag, pp. 37–61.
[8] Nielson, H. R., and Nielson, F. *Semantics with Applications: A Formal Introduction*. John Wiley & Sons, Inc., 2003.
[9] Nilsson, H., Ed. *Trends in Functional Programming*, vol. 7. Intellect Books, 2007.
[10] Planas, E., Cabot, J., and Gómez, C. Verifying action semantics specifications in uml behavioral models. In *Proceedings of the 21st International Conference on Advanced Information Systems Engineering* (Berlin, Heidelberg, 2009), CAiSE '09, Springer-Verlag, pp. 125–140.
[11] Ruei, R., and Slonneger, K. Semantic prototyping: Implementing action semantics in standard ML. The University of Iowa, 1993.
[12] Sedláček, J. *Úvod do teorie grafů*. Academia, Praha, 1981.
[13] Slodičák, V., and Macko, P. How to apply linear logic in coalgebraical approach of computing. In *Proceedings of the 22nd Central European Conference on Information and Intelligent Systems, September 21-23, 2011, Varaždin, Croatia* (2011), University of Zagreb, pp. 373–380. ISSN 1847-2001.

[14] Slodičák, V., and Macko, P. New approaches in functional programming using algebras and coalgebras. In *European Joint Conferrences on Theory and Practise of Software - ETAPS 2011* (March 2011), Workshop on Generative Technologies, Universität des Saarlandes, Saarbrücken, Germany, pp. pp. 13–23. ISBN 978-963-284-188-5.

[15] Slodičák, V., and Macko, P. Some new approaches in functional programming using algebras and coalgebras. *Electronic Notes in Theoretical Computer Science Vol. 279*, Issue 3 (2011), pp. 41–62.

[16] Stuurman, G. Action semantics applied to model driven engineering, November 2010. University of Twente.

[17] Watt, D. An action semantics of standard ml. *Lecture Notes in Computer Science Vol. 298/1988* (1988).

Faculty of Electrical Engineering and Informatics, Technical University of Košice, Slovak Republic

E-mail address: viliam.slodicak@tuke.sk, valerie.novitzka@tuke.sk

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVII**, Number 1, 2012

A PRIMAL-DUAL INTERIOR POINT ALGORITHM FOR CONVEX QUADRATIC PROGRAMS

MOHAMED ACHACHE AND MOUFIDA GOUTALI

Abstract. In this paper, we propose a feasible primal-dual path-following algorithm for convex quadratic programs. At each interior-point iteration the algorithm uses a full-Newton step and a suitable proximity measure for tracing approximately the central path. We show that the short-step algorithm has the best known iteration bound,namely $O(\sqrt{n}\log\frac{(n+1)}{\epsilon})$.

1. Introduction

Consider the quadratic program (PQ) in standard format:

$$(P)\quad \min_x \left\{ c^T x + \frac{1}{2} x^T Q x \ : Ax = b,\ x \geq 0 \right\}$$

and its dual problem

$$(D)\quad \max_{(x,\,y,\,z)} \left\{ b^T y - \frac{1}{2} x^T Q x : A^T y + z - Qx = c,\ z \geq 0 \right\}.$$

Here Q is a given $n \times n$ real symmetric matrix,A is a given $m \times n$ real matrix, $c \in \mathbb{R}^n, b \in \mathbb{R}^m, x \in \mathbb{R}^n, z \in \mathbb{R}^n$ and $y \in \mathbb{R}^m$. The vectors x, y, z are the vectors of variables.

Quadratic programs appear in many areas of applications. For example in finance (portfolio optimization) and also as subproblems in sequential Quadratic Programming [1,5,6,10,11]. Interior point methods (IPMs) are among the most effective methods to solve a large wide of optimization problems. Nowadays most popular and robust methods of them are primal-dual path-following algorithms due to their numerical efficiency and their theoretical polynomial complexity [1,2,6-11]. The success of these algorithms for solving linear optimization LO leads researchers to extend it naturally to other important problems such as semidefinite programs SDP, quadratic programs QP, complementarity CP and conic optimization problems COP.

Received by the editors: February 5, 2012.

2010 *Mathematics Subject Classification.* 90C25, 90C51.

Key words and phrases. Convex quadratic programs; interior point methods; primal-dual algorithms; complexity of algorithms.

In this paper, we propose a feasible short-step primal-dual interior point (IP) algorithm for convex quadratic programs. The algorithm uses at each interior-point iteration a full-Newton step and a suitable proximity measure for tracing approximately the central path. For its complexity analysis, we reconsider the analysis used by many researchers for LO and we make it suiting for CQP case. We show that the algorithm has the best known iteration bound $O(\sqrt{n}\log\frac{(n+1)}{\epsilon})$ which is analogous to LO.

The rest of the paper is organized as follows. In section 2, a feasible short-step primal-dual IP algorithm for CQP is presented. In section 3,the complexity analysis of the algorithm is discussed. In section 4, a conclusion is stated.

The notation used in this paper is: $\mathbb{R}^n$ denotes the space of n-dimensional real vectors. Given $x,\ y \in \mathbb{R}^n$, $x^T y = \sum_{i=1}^n x_i y_i$ is their usual scalar product whereas $xy = (x_1 y_1, \ldots, x_n y_n)^T$ is the vector of their coordinate-wise product. The standard 2-norm and the maximum norm for a vector x are denoted by $\|x\|$ and $\|x\|_\infty$, respectively. Let $x \in \mathbb{R}^n$, $\sqrt{x} = (\sqrt{x_1}, \ldots, \sqrt{x_n})^T$, $x^{-1} = (x_1^{-1}, \ldots, x_n^{-1})^T$ if $x_i \neq 0$ for all i and $(\frac{x}{y}) = (\frac{x_1}{y_1}, \ldots, \frac{x_n}{y_n})^T$ for $y_i \neq 0$. Let $g(t)$ and $f(t)$ be two positive real valued functions, then $g(t) = O(f(t)) \Leftrightarrow g(t) \leq cf(t)$ for some $c > 0$. The vector of ones in $\mathbb{R}^n$ is denoted by e .

2. A primal-dual path following IP algorithm for CQP

Throughout the paper, we make the following assumptions on *(P)* and *(D)*.

- The matrix A is of rank m (rg $(A) = m < n$).
- Interior-Point-Condition (IPC). There exists (x^0, y^0, z^0) such that

$$Ax^0 = b,\ A^T y^0 - Qx^0 + z^0 = c,\ x^0 > 0,\ z^0 > 0.$$

- The matrix Q is positive semidefinite,i.e., for all $x \in \mathbb{R}^n : x^T Q x \geq 0$.

It is well-known under our assumptions that solving *(P)* and *(D)* is equivalent to solve the Karush-Khun-Tucker optimality conditions for *(P)* and *(D)*:

$$\left\{\begin{array}{lcl} Ax & = & b,\ x \geq 0, \\ A^T y + z - Qx & = & c,\ z \geq 0, \\ xz & = & 0. \end{array}\right. \tag{1}$$

Now, by replacing the complementarity equation $xz = 0$ in (1) by the perturbed equation $xz = \mu e$, one obtains the following perturbed system:

$$\left\{\begin{array}{lcl} Ax & = & b,\ x \geq 0, \\ A^T y + z - Qx & = & c,\ z \geq 0, \\ xz & = & \mu e, \end{array}\right. \tag{2}$$

with $\mu > 0$. It is also known under our assumptions that the system (2) has a unique solution for each $\mu > 0$, denoted by $(x(\mu), y(\mu), z(\mu))$,we call $x(\mu)$ the $\mu-$center of *(P)* and $(y(\mu), z(\mu))$ the $\mu-$center of *(D)*. The set of $\mu-$centers

gives a homotopy path, which is called the central path of *(P)* and *(D)*. If μ goes to zero, then the limit of the central path exists and since the limit point satisfies the complementarity condition, the limit yields an optimal solution for *(P)* and *(D)*. The notion of the central path has been studied by many authors (see the books,e.g.[9] [10] and [11]).

Primal-dual path-following interior point algorithms are iterative methods which aim to trace approximately the central path by using at each interior iteration a feasible Newton step and get closer to a solution of (2) as μ goes to zero, e.g. [6-11].

Now, we proceed to describe a full-Newton step produced by the algorithm for a given $\mu > 0$. Applying Newton's method for (2) for a given feasible point (x, y, z) then the Newton direction $(\Delta x, \Delta y, \Delta z)$ at this point is the unique solution of the following linear system of equations:

$$\begin{cases} A\Delta x & = & 0, \\ A^T\Delta y + \Delta z - Q\Delta x & = & 0, \\ x\Delta z + z\Delta x & = & \mu e - xz. \end{cases}$$

This last can be written as:

$$\begin{pmatrix} A & 0 & 0 \\ -Q & A^T & I \\ Z & 0 & X \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta z \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \mu e - Xz \end{pmatrix}, \tag{3}$$

where $X :=\text{diag}(x)$, $Z :=\text{diag}(z)$ and I is the identity matrix of order n. The format in (3) is suitable for numerical implementation. Hence an update full-Newton step is given by $x_+ = x + \Delta x$, $y_+ = y + \Delta y$ and $z_+ = z + \Delta z$.
Now, we introduce a norm-based proximity as:

$$\delta(xz; \mu) = \frac{1}{2} \left\| \sqrt{\left(\frac{xz}{\mu}\right)^{-1}} - \sqrt{\frac{xz}{\mu}} \right\|,$$

to measure the closeness of feasible points to the central path. We use also a threshold value β and we suppose that a strictly feasible point (x^0, y^0, z^0) such that $\delta(x^0z^0; \mu^0) \leq \beta$ for certain μ^0, is known. The details of the algorithm are stated in the next subsection.

2.1. **Algorithm.**

Input
An accuracy parameter $\epsilon > 0$;
a threshold parameter $0 < \beta < 1$ (default $\beta = \frac{1}{\sqrt{2}}$);
a fixed barrier update parameter $0 < \theta < 1$ (default $\theta = \frac{1}{2\sqrt{n}}$);
a strictly feasible (x^0, y^0, z^0) and μ^0 such that $\delta(x^0 z^0; \mu^0) \leq \beta$;
begin
$x := x^0$; $y := y^0$; $z := z^0$; $\mu := \mu^0$;
while $n\mu \geq \epsilon$ **do**
begin
$\mu := (1 - \theta)\mu$;
solve the system (3) to obtain: $(\Delta x, \Delta y, \Delta z)$;
update $x = x + \Delta x$, $y = y + \Delta y$, $z = z + \Delta z$;
end
end

Fig.1. **Algorithm 2.1**

One distinguishes IPMs as short-step when $\theta = O\left(\frac{1}{\sqrt{n}}\right)$ and large-step when $\theta = O(1)$.

3. Complexity analysis

In this section, we discuss the complexity analysis of Algorithm 2.1. For convenience, we introduce the following notation. The vectors

$$v := \sqrt{\frac{xz}{\mu}}, \; d := \sqrt{\frac{x}{z}}.$$

The vector d is used to scale the vectors x and z to the same vector v :

$$\frac{d^{-1}x}{\sqrt{\mu}} = \frac{dz}{\sqrt{\mu}} = v,$$

as well as for the original directions Δx and Δz to

$$d_x := \frac{d^{-1}\Delta x}{\sqrt{\mu}}, \; d_z := \frac{d\Delta z}{\sqrt{\mu}}.$$

In addition, we have

$$x\Delta z + z\Delta x = \mu v(d_x + d_z), \tag{4}$$

and

$$\Delta x \Delta z = \mu d_x d_z. \tag{5}$$

By using these notations the linear system in (3) and the proximity become:

$$\left\{\begin{array}{c} \bar{A}\, d_x = 0 \\ \bar{A}^{\,T}\Delta y + d_z - \bar{Q}\, d_x = 0 \\ d_x + d_z = v^{-1} - v, \end{array}\right. \tag{6}$$

where $\bar{A} = \sqrt{\mu}AD$ and $\bar{Q} = \sqrt{\mu}DQD$ with $D := \mathrm{diag}\,(d)$ and

$$\delta(xz;\mu) := \delta(v) = \frac{1}{2}\|v^{-1} - v\|.$$

Now observe that

$$d_x^T d_z = d_x^T \bar{Q}\, d_x \geq 0,$$

since

$$d_x^T d_z = d_x^T(\bar{Q}d_x - \bar{A}^T d_y) = d_x^T \bar{Q} d_x \geq 0,$$

with $\bar{A}d_x = 0$ and $\bar{Q}$ is positive semidefinite.
This last inequality shows that Δx and Δz are not orthogonal directions in contrast with LO case. Thus makes the analysis of the algorithm for CQP more difficult.

Next, we need the following technical lemma that will be used later in the analysis of the algorithm. For its proof the reader can refer to [Lemma C.4 in 8].

Lemma 3.1. *Let (d_x, d_z) be a solution of (6) and if $\delta := \delta(xz;\mu)$ and $\mu > 0$. Then one has*

$$0 \leq d_x^T d_z \leq 2\delta^2 \tag{7}$$

and

$$\|d_x d_z\|_\infty \leq \delta^2, \quad \|d_x d_z\| \leq \sqrt{2}\delta^2\,. \tag{8}$$

In the next lemmas, we state conditions which ensure the strict feasibility of a full-Newton step.

Lemma 3.2. *Let (x, z) be a strictly feasible primal-dual point. If*

$$e + d_x d_z > 0,$$

then $x_+ = x + \Delta x > 0$ and $z_+ = z + \Delta z > 0$.

Proof. To show that x_+ and z_+ are positive, we introduce a step length $\alpha \in [0, 1]$ and we define

$$x^\alpha = x + \alpha\Delta x, \quad z^\alpha = z + \alpha\Delta z.$$

So $x^0 = x, x^1 = x_+$ and similar notations for z, hence $x^0 z^0 = xz > 0$. We have

$$x^\alpha z^\alpha = (x + \alpha\Delta x)(z + \alpha\Delta z) = xz + \alpha(x\Delta z + z\Delta x) + \alpha^2\Delta x\Delta z.$$

Now by using (4),and (5) we get

$$x^\alpha z^\alpha = xz + \alpha(\mu e - xz) + \alpha^2 \Delta x \Delta z.$$

We assume that $e + d_x d_z > 0$, we deduce that $\mu e + \Delta x \Delta z > 0$ which is equivalent to $\Delta x \Delta z > -\mu e$. By substitution we obtain

$$\begin{aligned} x^\alpha z^\alpha &> xz + \alpha(\mu e - xz) - \alpha^2 \mu e \\ &= (1-\alpha)xz + (\alpha - \alpha^2)\mu e \\ &= (1-\alpha)xz + \alpha(1-\alpha)\mu e. \end{aligned}$$

Since xz and μe are positive it follows that $x^\alpha z^\alpha > 0$ for $\alpha \in [0,1]$. Hence, none of the entries of x^α and z^α vanish for $\alpha \in [0,1]$. Since x^0 and z^0 are positive, this implies that $x^\alpha > 0$ and $z^\alpha > 0$ for $\alpha \in [0,1]$. Hence, by continuity the vectors x^1 and z^1 must be positive which proves that x_+ and z_+ are positive. This completes the proof. □

Now for convenience, we may write

$$v_+^2 = \frac{x_+ z_+}{\mu}$$

and it is easy to have

$$v_+^2 = e + d_x d_z.$$

Lemma 3.3. *If $\delta := \delta(x,z;\mu) < 1$. Then $x_+ > 0$ and $z_+ > 0$.*

Proof. We have seen in Lemma 3.2, that $x_+ > 0$ and $z_+ > 0$ are strictly feasible if $e + d_x d_z > 0$. So $e + d_x d_z > 0$ holds if $1 + (d_x d_z)_i > 0$ for all i. In fact we have

$$\begin{aligned} 1 + (d_x d_z)_i &\geq 1 - |(d_x d_z)_i| \quad \text{for all } i, \\ &\geq 1 - \|d_x d_z\|_\infty \end{aligned}$$

and by the bound in (9) it follows that

$$1 + (d_x d_z)_i \geq 1 - \delta^2.$$

Thus $e + d_x d_z > 0$ if $\delta < 1$. This completes the proof. □

The next lemma shows the influence of a full-Newton step on the proximity measure.

Lemma 3.4. *If $\delta(xz;\mu) < 1$. Then*

$$\delta_+ := \delta(x_+ z_+, \mu) \leq \frac{\delta^2}{\sqrt{2(1-\delta^2)}}.$$

Proof. We have

$$\begin{aligned} 4\delta_+^2 &= \left\|v_+^{-1} - v_+\right\|^2 \\ &= \left\|v_+^{-1}(e - v_+^2)\right\|^2. \end{aligned}$$

But $v_+^2 = e + d_x d_z$ and $v_+^{-1} = \frac{1}{\sqrt{e+d_x d_z}}$, then it follows that

$$4\delta_+^2 = \left\| \frac{d_x d_z}{\sqrt{e + d_x d_z}} \right\|^2 \leq \frac{\|d_x d_z\|^2}{1 - \|d_x d_z\|_\infty}.$$

Now in view of Lemma 3.1, we deduce that $4\delta_+^2 \leq \frac{2\delta^4}{1-\delta^2}$.
This completes the proof. □

Corollary 3.1. *If $\delta := \delta(xz;\mu) < \frac{1}{\sqrt{2}}$. Then $\delta_+ \leq \delta^2$ which means the quadratic convergence of the proximity measure during a full-Newton step.*

In the next lemma, we discuss the influence on the proximity measure of an update barrier parameter $\mu_+ = (1-\theta)\,\mu$ during the Newton process along the central path.

Lemma 3.5. *If $\delta(xz;\mu) < \frac{1}{\sqrt{2}}$ and $\mu_+ = (1-\theta)\,\mu$ where $0 < \theta < 1$. Then*

$$\delta^2(x_+ z_+;\mu_+) \leq (1-\theta)\delta_+^2 + \frac{\theta^2(n+1)}{4(1-\theta)} + \frac{\theta}{2}.$$

In addition if $\delta \leq \frac{1}{\sqrt{2}}$, $\theta = \frac{1}{2\sqrt{n}}$ and $n \geq 2$, then we have:

$$\delta(x_+ z_+;\mu_+) \leq \frac{1}{\sqrt{2}}.$$

Proof. Let $v_+ = \sqrt{\frac{x_+z_+}{\mu}}$ and $\mu_+ = (1-\theta)\,\mu$. Then

$$\begin{aligned}
4\delta^2(x_+z_+;\mu_+) &= \left\| \left(\sqrt{\frac{\mu_+}{x_+z_+}}\right) - \left(\sqrt{\frac{x_+z_+}{\mu_+}}\right) \right\|^2 \\
&= \left\| \sqrt{1-\theta} v_+^{-1} - \frac{1}{\sqrt{1-\theta}} v_+ \right\|^2 \\
&= \left\| \sqrt{1-\theta}(v_+^{-1} - v_+) - \frac{\theta}{\sqrt{1-\theta}} v_+ \right\|^2 \\
&= (1-\theta)\left\|v_+^{-1} - v_+\right\|^2 + \frac{\theta^2}{1-\theta}\left\|v_+\right\|^2 - 2\theta(v_+^{-1} - v_+)^T v_+ \\
&= (1-\theta)\left\|v_+^{-1} - v_+\right\|^2 + \frac{\theta^2}{1-\theta}\left\|v_+\right\|^2 - 2\theta(v_+^{-1})^T v_+ + v_+^T v_+ \\
&= 4(1-\theta)\delta_+^2 + \frac{\theta^2}{1-\theta}\left\|v_+\right\|^2 - 2\theta n + 2\theta\left\|v_+\right\|^2 \ ,
\end{aligned}$$

since $(v_+^{-1})^T v_+ = n$ and ${v_+}^T v_+ = \|v_+\|^2$. Now, recall that $\delta_+^2 \leq \frac{\delta^4}{2(1-\delta^2)}$. Then

$$4\delta^2(x_+z_+;\mu_+) \leq 4(1-\theta)\frac{\delta^4}{2(1-\delta^2)} + \frac{\theta^2}{1-\theta}\left\|v_+\right\|^2 - 2\theta n + 2\theta\left\|v_+\right\|^2.$$

Finally, since

$$x_+^T z_+ = \mu n + \mu d_x^T d_z,$$

and if $\delta < \frac{1}{\sqrt{2}}$, it follows by (8) in Lemma 3.1 that

$$\|v_+\|^2 = \frac{1}{\mu} x_+^T z_+ \leq (n+1),$$

and consequently

$$4\delta^2(x_+z_+,\mu_+) \leq 4(1-\theta)\frac{\delta^4}{2(1-\delta^2)} + \frac{\theta^2(n+1)}{1-\theta} - 2\theta n + 2\theta(n+1)$$

and

$$\delta^2(x_+z_+;\mu_+) \leq (1-\theta)\delta_+^2 + \frac{\theta^2(n+1)}{4(1-\theta)} + \frac{\theta}{2}.$$

For the last statement the proof goes as follows. If $\delta < \frac{1}{\sqrt{2}}$, then $\delta_+^2 < \frac{1}{4}$ and this yields the following upper bound for $\delta^2(x_+z_+;\mu_+)$ as:

$$\delta^2(x_+z_+;\mu_+) \leq \frac{(1-\theta)}{4} + \frac{\theta^2(n+1)}{4(1-\theta)} + \frac{\theta}{2}.$$

Now, taking $\theta = \frac{1}{2\sqrt{n}}$ then $\theta^2 = \frac{1}{4n}$, it follows that:

$$\delta^2(x_+z_+;\mu_+) \leq \frac{\frac{1}{4n}(n+1)}{4(1-\theta)} + \frac{\theta}{2} + \frac{(1-\theta)}{4},$$

and since $\frac{n+1}{4n} \leq \frac{3}{8}$ for all $n \geq 2$,then we have:

$$\delta^2(x_+z_+;\mu_+) \leq \frac{3}{32(1-\theta)} + \frac{\theta}{2} + \frac{(1-\theta)}{4}.$$

Now for $n \geq 2$,we have $0 \leq \theta \leq \frac{1}{2\sqrt{2}}$ and since the function

$$f(\theta) = \frac{3}{32(1-\theta)} + \frac{(1-\theta)}{4} + \frac{\theta}{2}$$

is continuous and monotonic increasing on $0 < \theta < \frac{1}{2\sqrt{2}}$, consequently

$$f(\theta) \leq f\left(\frac{1}{2\sqrt{2}}\right) \approx 0.48341 < \frac{1}{2}, \text{for all } \theta \in \left[0, \frac{1}{2\sqrt{2}}\right].$$

Hence $\delta(x_+z_+;\mu_+) < \frac{1}{\sqrt{2}}$ and the algorithm is well-defined. This completes the proof.
In the next lemma we analyze the effect of a full-Newton step on the duality gap.

Lemma 3.6. *Let $\delta := \delta(xz;\mu) < \frac{1}{\sqrt{2}}$ and $x_+ = x + \Delta x$ and $z_+ = z + \Delta z$. Then the duality gap satisfies*

$$x_+^T z_+ \leq \mu(n+1). \tag{9}$$

Proof. It follows straightforwardly from the proof in Lemma 3.5. □

In the next lemma we compute a bound for the number of iterations of Algorithm 2.1.

Lemma 3.7. *Let x^{k+1} and z^{k+1} be the $(k+1)$-th iteration produced by Algorithm 2.1 with $\mu := \mu_k$. Then*

$$(x^{k+1})^T z^{k+1} \leq \epsilon$$

if

$$k \geq \left\lceil \frac{1}{\theta} \log \frac{\mu_0(n+1)}{\epsilon} \right\rceil.$$

Proof. It follows form the bound(10) in Lemma 3.6 that:

$$(x^{k+1})^T z^{k+1} \leq \mu_k(n+1)$$

with

$$\mu_k = (1-\theta)\mu_{k-1} = (1-\theta)^k \mu_0.$$

Then it follows that:

$$(x^{k+1})^T z^{k+1} \leq (1-\theta)^k \mu_0 (n+1).$$

Thus the inequality $(x^{k+1})^T z^{k+1} \leq \epsilon$ is satisfied if

$$(1-\theta)^k \mu_0 (n+1) \leq \epsilon.$$

Now taking logarithms of $(1-\theta)^k \mu_0 (n+1) \leq \epsilon$, we may write

$$k \log(1-\theta) \leq \log \epsilon - \log \mu_0 (n+1)$$

and using the fact that $\log(1-\theta) \leq \theta$, for $0 \leq \theta < 1$ then the above inequality holds if

$$k\theta \geq \log \mu_0 (n+1) - \log \epsilon = \log \frac{(n+1)\mu_0}{\epsilon}.$$

This completes the proof. □

For $\theta = \frac{1}{2\sqrt{n}}$, we obtain the following theorem

Theorem 3.1. *Let $\theta = \frac{1}{2\sqrt{n}}$. Then Algorithm 2.1 requires at most*

$$O\left(\sqrt{n} \log \frac{(n+1)\mu^0}{\epsilon}\right)$$

iterations.

Proof. By replacing $\theta = \frac{1}{2\sqrt{n}}$ in Lemma 3.7, the result holds. □

4. Conclusion and future works

In this paper, we have proposed a feasible short-step primal-dual interior point algorithm for solving CQP. The algorithm deserves the best well-known theoretical iteration bound $O(\sqrt{n} \log \frac{(n+1)}{\epsilon})$ when the starting point is $x_0 = z_0 = e$. This choice of initial point can be done by the technique of embedding. Future research might extended the algorithm for other optimization problems and its numerical implementation is also an interesting topic.

References

[1] M. Achache. A new primal-dual path-following method for convex quadratic programming. Comput. Appl. Math., 25 pp. 97 -110 (2006).

[2] M. Achache. A weighted path-following method for the linear complementarity problem. Universitatis Babes. Bolyai. Series Informatica (49) (1) pp. 61-73 (2004).

[3] M. Achache. Complexity analysis and numerical implementation of a short step primal-dual algorithm for linear complementarity problem. Applied Mathematics and computation. (216),1889-1895 (2010).

[4] L. Adler and R.D.C. Monteiro. Interior path-following primal-dual algorithms. Part II: convex quadratic programming. Mathematical programming,(44),43-66. (1989).

[5] A. Ben-Tal and A. Nemirovski. Lectures on Modern convex optimization. Analysis, Algorithms and engineering Applications, volume 2 of MPS-SIAM. Series on optimization. SIAM. Philadelphia.. USA (2000).
[6] Zs. Darvay. New interior-point algorithms in linear optimization [J]. Advanced Modeling and optimization.5(1):51-92.(2003).
[7] B. Jansen,C. Roos,T. Terlaky and J.Ph. Vial. Primal-dual target-following algorithms for linear programing. Report 93-107. Delf University of technology. (1993).
[8] J. Peng, C. Roos and T.Terlaky. New complexity analysis of the primal-dual Newton method for linear optimization. Annals of operations research.(49) pp.23-39.(2000).
[9] C. Roos, T.Terlaky, and J.Ph. Vial. Theory and algorithms for linear optimization. An interior point approach. John-Wiley. Sons,Chichester, UK,1997.
[10] S.J. Wright. Primal-dual interior point methods. SIAM,Philadelphia, USA.(1997).
[11] Y.Ye. Interior point algorithms. Theory and Analysis. John-Wiley. Sons, Chichester, UK,1997.

Laboratoire de Mathématiques Fondamentales et Numériques. Université Ferhat Abbas de Sétif. Algérie
E-mail address: achache_m@yahoo.fr

Département de Mathématiques. Faculté des Sciences. Université Ferhat Abbas de Sétif. Algérie
E-mail address: moufida_g2006@yahoo.fr

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVII**, Number 1, 2012

A PROPOSED DSL FOR DATA INTENSIVE APPLICATION DEVELOPMENT

PAUL HORAŢIU STAN

Abstract. Model Driven Architecture (MDA) defines three layers of abstraction for a domain: Computation Independent Model (CIM), Platform Independent Model (PIM) and Platform Specific Model (PSM). Nowadays in software industry the translation from PIM to PSM is made by developers that implement the model diagrams. This paper presents a new Domain Specific Language (DSL) for developing data intensive applications. The proposed DSL contains a grammar for specifying the PIM and a transformation engine to .NET PSM. The benefits of the proposed DSL resides in the possibility to write transformations to many PSM not only to .NET web applications. Finally a comparison with WebML and WebDSL is presented.

1. Introduction

In software engineering, a DSL represents a custom programming language dedicated to a particular problem domain. It contains a minimal set of statements understood by the people who use domain specific concepts.[11].

This paper presents a DSL for specifying data intensive web applications. Using the proposed DSL, a PIM of an application can be defined, then using transformations to given PSM the source code can be automatically generated.

The terms PIM and PSM are most frequently used in the context of the MDA [1] approach. This MDA approach corresponds to the OMG vision of Model Driven Engineering. The main idea is that it should be possible to use a Model Transformation Language (MTL) to transform a PIM into a PSM.

The paper is structured as follows: Section 2 presents the problem that the current research intend to solve, divided into two subsections that describe (1) current DSL for data intensive web applications and (2) the proposed solution;

Received by the editors: September 26 2011.

2010 *Mathematics Subject Classification.* 68N15, 68N20.

1998 *CR Categories and Descriptors.* D.2.11 [**Software**]: SOFTWARE ENGINEERING – *Software Architectures* D.2.13 [**Software**]: SOFTWARE ENGINEERING – *Reusable Software*; D.3.4 [**Software**]: PROGRAMMING LANGUAGES – *Processors*;

Key words and phrases. Domain Specific Languages, Model Transformation, Code generators.

Section 3 shows the technical details of the proposed DSL for data intensive applications together with a comparison with WebML and WebDSL two of the most actual DSLs for web modeling, and finally, Section 4 summarizes the research results.

2. The Problem

In order to improve the quality and the development time of the final software products, the software engineering industry should use abstraction more and more. By retaining knowledge about lower level operations in higher-level abstractions, developers can work with higher-level concepts and save the effort of composing the lower-level operations [11, 10].

The conventional abstraction technique that involves methods and classes are no longer sufficient for creating new abstraction layers [15, 1]. Libraries and frameworks are good at encapsulating functionality, but it is often awkward for developers to reach that functionality, using, in many cases the application programmers interface (API) [11].

The common parts of the domain are implemented by code generation templates, while the custom variables are configurated by the application developer using configuration interfaces. These configuration interfaces can take the form of a wizard for simple domains, or complex languages for larger domains [11].

The scope of our research is web applications with a rich data model. That is, applications with a database for data storage and a user interface providing several views on the data in the database, including CRUD operations. An additional assumption is that the data model cannot be changed at run-time.

2.1. Current DSLs for web applications.

There are many development environments and programming languages that can be used to design and implement data intensive applications, but these are general purpose languages, in other words, are not specialized only on a given domain.

This section presents two of the actual DSLs for web application development: Web Modeling Language (WebML) and Web Domain Specific Language (WebDSL).

Web Modeling Language (WebML) can be used to define web sites under distinct dimensions: (1) structural model which expresses the data content of the site, in terms of the entities and relationships, (2) composition model contains the pages that compose it, (3) navigation model represents the topology of links between pages, (4) presentation model defines the layout and graphic requirements for page rendering, (5) personalization model contains the customization features for one-to-one content delivery. All the concepts of WebML have two representations: graphic notation and a textual XML

syntax. WebML specifications are independent of programming languages or development platforms. WebML is a model-driven approach to web site development [6, 14]. WebML enables developers to define the core features of a web application at a higher level avoiding architectural details. All proposed concepts are associated with intuitive graphical symbols which can be easily supported by CASE tools and useful for the non-technical members of the application development team [16, 14].

WebML defines basic units such as: Data unit, Index unit, Entry unit, Create unit, Delete unit, etc that have graphical representation and a default implementation; for instance the Create unit enables the creation of a new entity instance [16, 2]. These basic units are translated to a PSM in order to become an application. One of the commercial tools that implement WebML specifications is WebRatio.

WebDSL [8, 3, 9, 4] is another DSL for developing web applications with a data model. The main features of WebDSL are: (1) Domain modeling, (2) Presentation, (3) Page-flow, (4) Access control, (5) Data validation, (6) Workflow, (7) Styling, (8) Email. WebDSL applications are translated to Java web applications, and the code generator is implemented using Stratego/XT and SDF [11, 10].

WebDSL has the following types of statements: (1) function definition, (2) variable declaration, (3) assignment, (4) if, (5) return, (6) for loop, (7) while and (8) switch [4].

The definition of a page has three parts: (1) the name of the page, (2) parameters definition, and (3) a presentation of the data encapsulated in the parameters. WebDSL provides basic markup operators such as: (1) section, (2) header, and (3) list for defining the structural model of a page. Data from the object parameters are displayed in the page by data access operations such as `output`. Collections of data can be iterated using the `for` construct. It is possible to display the content of an object based on a condition. Using custom templates the developer can define reusable parts of code. Finally, WebDSL supports the separation of the concepts, user-defined concepts can be grouped in modules [4].

2.2. **The proposed DSL for web applications.** This section presents the current DSLs for data intensive web applications and the big picture of the proposed DSL for developing data intensive applications.

2.2.1. *Problem Motivation.* DSLs for web applications intend to solve the same problem, all of them are focus on web applications domains and intend to provide a common language for these domains. A new DSL should contain fewer statements than a general-programming language like Java or .NET and should work with more abstract things and should be able to (1) automatically

transform a PIM into a general-programming language and to (2) simulate the PIM before the transformation process.

At this moment, WebML and WebDSL have transformation engines only to Java. The main motivations of the proposed DSL are: the PIM should be easily translated to different programming languages, the syntax should contains fewer "words" than WebDSL, the language should be easily extended with platform dependent routines.

2.2.2. *Conceptual view of the proposed solution.* Nowadays many applications process data, these data are shown to the user via windows, web pages, different type of mobile forms etc. In the proposed solution these things are modeled using the *Page* concept. A page shows/reads data to/from the user interface and receives user's actions. For displaying and reading data few statements are needed, and for catching user actions the event concept is introduced into the proposed solution. So, it is a good thing for each defined event to have an event handler, the body of the event handler contains a set of statements that control the application flow. In the proposed solution these statements are structured into four categories: (1) User Interface Statements, (2) Domain Statements, (3) Persistence Statements and (4) Control Flow Statements. They will be detailed later into this paper. The statements work with domain objects: (1) Entities, (2) Value Objects and (3) Specifications. Domain objects are used to define the static model of the application, while statements define its dynamic model.

3. Technical details

This section describes the proposed solution's details. First subsection shows the architecture of the proposed DSL, the second describes how the solution works in a real context and the last one makes a comparison with WebML and WebDSL during the development process of a web site that manages books, authors and members.

3.1. **The proposed DSL's architecture.** Figure 1 shows the metamodel of the proposed DSL for developing data intensive applications. The solution involves (1) a simple language with few concepts to ensure the syntax will be easily understandable by developers and (2) a transformation engine to different PSMs. At this moment only the transformation to .NET web sites is available, but the PIM allows transformations to Java web sites, .NET desktop applications, Java desktop applications, Java and .NET mobile applications, etc.

The metamodel classes are divided into three main categories:

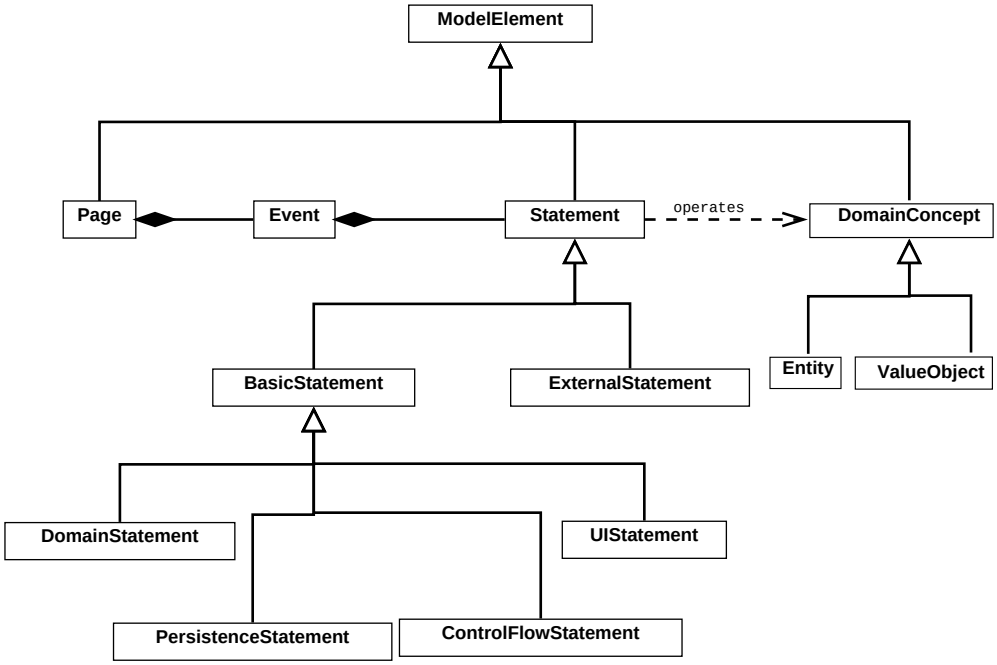


FIGURE 1. Proposed DSL's metamodel

- *Pages* and *Events*, used in order to manage the interaction with the human user, display data and read user actions;
- *Domain Concepts*, used for defining the business model of the system, that includes entities and relations between them;
- *Statements*, used for specifying the behavior of the system. There are *basic statements* and *external statements* the latter representing custom operations defined by the development team. An external statement is a link between a new platform-independent statement and a platform-dependent statement, acting as a wrapper.

Each page has a list of events with at least one event called *main event*, raised when the page is loaded. An event other than the *main event* represents a user action. Each event has an *event handler* with the same name. The *event handler* has a list of statements for describing the system response when a given event appears.

In order to define the static model of an application a developer can use the following domain concepts:

- *Entities*, can be stored into the persistent storage, can have properties of primitive types and can be associated with another entities in one

of the following ways: one to one, one to many, many to one and many to many. The Domain Driven Design (DDD) patterns have been used for designing the *Entity* metamodel class [12].

- *Value Objects*, which cannot be stored into the persistent storage, can have only properties of primitive types and cannot be associated with another domain concept. The name *Value Object* has been adapted from [12].
- *Specification*, validation rules for *Entities* or *Value Objects*, containing a list of properties like an entity and a logical expression that will be evaluated for a given domain object. Available operators for composing the logical expression are: *and, or, not, not=, <=, >=, <, >, ==, -, +, *, /*.

Listing 1 presents a specification example. The name of the specification is *FilterMembersByName* and it is applicable to *Member* entities. The specification has a single property of type *String* called *myMemberName*. The logical expression checks if the first name or the last name of a given member contains the *myMemberName* property as a substring. The logical expression can be composed with logical operators and with *specification functions* which are either (1) Basic functions, defined into the proposed DSL or (2) External functions being like external statements links between new platform-independent functions and platform-specific functions implemented by the development team.

Listing 1. A specification example

```
1 Specification FilterMembersByName for Member{
2   myMemberName:String
3   expression: (contains(Member.FName,myMemberName) or contains
        (Member.LName,myMemberName))
4 }
```

For defining user interfaces and system behavior, the developer should create pages and enter control statements in the bodies of event handlers for controlling the processing flow.

The basic statements are divided into four main categories:

- *Domain Statements*, which operate over domain concepts such as *Entities*, *Value Objects* and *Specifications*.
- *User Interface Statements*, are used to display data to the user or to read data from the pages.
- *Persistence Statements*, useful when storing and loading entities to, respectively from, the repository.

- *Control flow statements*, a subset of similar statements from a general programming language.

3.2. **The transformation process.** The Transformation engine has two inputs: (1) the model specified in the proposed DSL and (2) platform-specific templates, in our case .NET templates. These templates contain code snippets for .NET. The result of the process is the generated source code of the application which is the PSM.

3.3. **How it works.** This section describes how the entire solution works, presents simple code examples written in the proposed DSL and the transformation process from the DSL code to the Microsoft C#.NET code. Listing 2 presents the essential elements of the proposed language in the Xtext grammar specification language.

Listing 2. DevDSL's grammar

```
/*Level 1*/
Model:{Model}
   'config' '{'Config+=Config*'}'
   elements+=AbstractElement*;

/*Level 2*/
AbstractElement: DomainObject | Specification | Page;
DomainObject: Entity | ValueObject;

/*Level 3*/
Entity: 'Entity ' name=ID '{'
  (attributes+=Attribute)*
'}';

ValueObject:'ValueObject' name=ID'{'
  (attributes+=ValueObjectAttribute)+
'}';

Specification: 'Specification 'name=ID'
     for '(domainObject=[DomainObject] | 'Object')'{'
  (attributes+=Attribute)*
    'expression'':'expression=LogicalExpression
'}';

Page: 'Page ' name=ID '{'
  FirstEvent=Event
  NextEvents+=Event*
'}';
```

```
29
30 /*Level 4*/
31 Attribute: name=ID ':' type=(RefType|StdType);
32 ValueObjectAttribute:name=ID':'type=StdType;
33
34 Config: name=QualifiedName '=' value=STRING;
35
36 Statement:
37 //Domain Statements
38 CreateObject|EraseObject|SetObjectProperty|GetObjectProperty|
       AddElementToList|RemoveElementFromList|Satisfy|Filter|
39 //UI Statements
40 DisplayObject|DisplayList|DisplayButton|ReadObjectFromUI|
       Redirect|
41 //DB Statements
42 SaveEntityToRepository|LoadList|DeleteEntityFromRepository|
43 //ControlFlow Statements
44 IfStatement|WhileStatement|ForEachStatement |
45 //External statements
46 ExternalStatement;
47 ...
```

In order to create a PIM of an application, a developer should write a text file named `[filename].app` which should contain the following sections:

- the *config* section
- the *domain objects* section
- the *pages* section.

Each application should start with the *config* section, which can contain database connection properties for instance hibernate configurations.

Listing 3. Config Section

```
1  config {
2    proxyfactory.factory_class=
3      "NHibernate.ByteCode.Castle.ProxyFactoryFactory,
4      NHibernate.ByteCode.Castle"
5    connection.provider=
6      "NHibernate.Connection.DriverConnectionProvider"
7    dialect=
8      "NHibernate.Dialect.MsSql2005Dialect"
9    connection.driver_class=
10     "NHibernate.Driver.SqlClientDriver"
11   connection.connection_string=
12     "Server=servername;Initial Catalog=database;
```

```
13      User Id=user;Password=*****"
14    hbm2ddl.auto="update"
15  }
```

The *domain objects* section contains definitions for *Entities*, *Value Objects* and *Specifications*, the static model of the application. *Entities* and *Value Objects* are similar to classes in a general object oriented programming language.

Listing 4. Domain Objects Section

```
1  Entity Book{                          Entity Author{
2    Title: String                         FName:String
3    ISBN : String                         LName:String
4    Price: Double                         BDate:String
5    Pages: Integer                        Royalties:Royalty*
6    Royalties:Royalty*                  }
7    LibraryBooks:LibraryBook*
8  }
9  Entity LibraryBook {                  Entity Royalty{
10   RealBook:Book                         RBook:Book
11   StockId:String                        RAuthor:Author
12   Value:Double                          RAmount:Double
13   Lends:Lend*                         }
14 }
15 Entity Member {                       Entity Lend {
16   FName:String                          LMember:Member
17   LName:String                          LLibraryBook:LibraryBook
18   BDate:String                          LendDate:String
19   Sex:String                            ReturnDate:String
20   Address:String                        Days:Integer
21   StartDate:String                      Fee:Double
22   Dues:Double                         }
23   Lends:Lend*
24 }
```

The listing 4 defines the same model as the figure 2 which is an WebML data model diagram.

The last section a developer should write is the *Pages* section, which contains the dynamic model of the application. The listing 5 presents a demo page specification using the proposed DSL.

Listing 5. Page Demo

```
1  Page CurrentBook{
2    Page_Load{
```

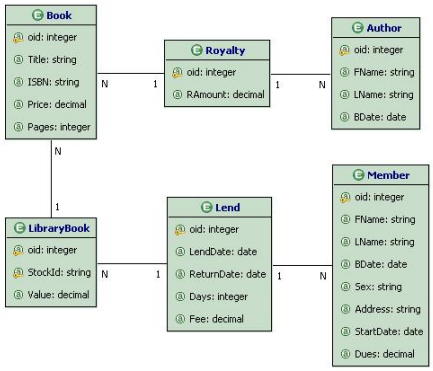


FIGURE 2. The Library Model

```
3     if(Satisfy("cBook",ObjectNotNull()))
4     {
5       DisplayObject("cBook",editable=true,
6         [("Library Books", btnLibraryBooks)
7          ("Royalties",btnRoyalties)])
8       DisplayButton("Back",btnBack)
9       DisplayButton("Save",btnSave)
10    }
11    else
12    {
13      Redirect(MainPage)
14    }
15  }
16  btnLibraryBooks{
17    ReadObjectFromUI("cBook")
18    Redirect(ManageLibraryBooks)
19  }
20  btnRoyalties{
21    ReadObjectFromUI("cBook")
22    Redirect(ManageRoyalties)
23  }
24  btnBack{
25    EraseObject("cBook")
26    Redirect(ManageBooks)
27  }
28  btnSave{
29    ReadObjectFromUI("cBook")
30    SaveEntityToRepository("cBook")
31    EraseObject("cBook")
```

```
32      Redirect(ManageBooks)
33    }
34  }
```

3.4. **The Library web site.** This section presents fragments from a demo web site called *Library* that manages books, authors and members, and has been developed using the proposed DSL. Also, parts of it have been developed using WebML and WebDSL in order to be able to make a comparison between our DSL and the others.

The conceptual model of the web site has been presented in the figure 2 for WebML and in the listing 6 for WebDSL. The conceptual model designed using our DSL is presented into the listing 4.

Listing 6. Library Model in WebDSL

```
entity Book{
  Title:: String
  ISBN :: String
  Price:: Double
  Pages:: Int
  Royalties -> Set<Royalty>
  LibraryBooks -> Set<LibraryBook>
}
entity LibraryBook {
  RealBook -> Book
  StockId::String
  Value::Double
  Lends -> Set<Lend>
}
entity Member {
  FName::String
  LName::String
  BDate::String
  Sex::String
  Address::String
  StartDate::DateTime
  Dues::Double
  Lends -> Set<Lend>
}

entity Author{
  FName::String
  LName::String
  BDate::String
  Royalties -> Set<Royalty>
}

entity Royalty{
  RBook -> Book
  RAuthor -> Author
  RAmount::Double
}

entity Lend {
  LMember::Member
  LLibraryBook->LibraryBook
  LendDate::DateTime
  ReturnDate::DateTime
  Days::Int
  Fee::Double
}
```

WebDSL offers the possibility to specify the kind of a relationship: (1) reference −> and (2) composite <>. The difference between reference and composite property kinds is that composite indicates that the referred entity is

part of the one referring to it. Deletion of the composite entity is also deleting the referred entities.

WebML uses graphic diagrams for defining the static model of an application. Unfortunately this graphic representation can confuse the developers because it is not similar to UML. For instance the relationship *one to many* in WebML is like the *many to one* in UML.

The proposed DSL does not offer the possibility to specify the association between two entities as reference or composite like WebDSL, but this can be added in a future version. In contrast with WebML, the proposed solution does not produce confusions regarding associations between entities. Entities are defined as classes in a GPL. Attributes of an entity are typed inside it's body, if an attribute is a collection of elements then after the attribute type there is an $*$.

The behavioral model of the library web site can be expressed in WebDSL using action code. Functions can be defined globally and as methods in entities. At this moment WebDSL model of an application can be translated only to Java PSM.

The behavioral model of the library web site can be expressed in WebML using units and links between them. The disadvantage of the WebML is that it is very hard to maintain a model with many lines and boxes; for complex sites, the model becomes a spaghetti picture. Current version of WebML supports only transformations to Java. On the other hand, if new units need to be defined, the developer should write the Java code which implements the unit functionality.

The dynamic model specified with the proposed DSL is easily understandable and manageable. At this moment the eclipse plug-in works with the textual models but, as a feature improvement, a visual plug-in can be developed. Unlike WebML, the model for complex systems, expressed using the proposed textual DSL, does not become an unmanageable model. Unlike WebDSL the proposed DSL does not allow developers to define methods inside the body of an entity. This fact ensures that the static and dynamic models are not mixed.

An advantage of our DSL compared with WebML and WebDSL is that it has a simple syntax with few "words". The conceps presented in the proposed DSL are structured in two main categories: concepts used to define static model: *Entities*, *Value Objects* and *Specifications*, and conceps used to define the behaviour of the system: *Pages*, *Events* and *Statements*. Due to this simple syntax, it is easy for a developer to understand the language and to quiqly ramp-up in an open project.

In contrast with both WebML and WebDSL, the PIMs described using the proposed DSL can be easily translated to different PSMs such as .NET, Java,

PHP, etc. It is in progress an implementation of a transformation engine to PHP PSM of the models specified using the proposed DSL.

This is a short comparison of the proposed DSL with WebML and WebDSL, the main target of this paper is to introduce the basic concepts of the new DSL, a more complex comparison can be the subject of a future paper.

4. Conclusions and future work

The main contributions of this paper are: (1) a new DSL for data intensive applications, (2) a textual representation of the proposed language and (3) a transformation engine from the proposed PIM to .NET web application PSM.

The novelty of the proposed solution resides in: (1) proposing a set of basic statements that can be easily translated to PSMs, (2) offering the possibility to add and call external statements, (3) introducing the concept of specification for defining validation rules for entities and value objects and (4) offering the possibility to translate the PIM to different PSMs such as web, desktop and mobile applications, all of these written in different programming languages.

The limitations of the proposed solution are: (1) it does not offer the inheritance relationship between entities, (2) it is not possible to specify if an association is container or reference, (3) a developer is not able to compose statements into parent statements for modularizing the system.

All these will be subject of future developments.

5. Acknowledgment

The author wish to thank for the financial support provided from programs co-financed by the SECTORAL OPERATIONAL PROGRAMME HUMAN RESOURCES DEVELOPMENT, Contract **POSDRU 6/1.5/S/3** "Doctoral studies: through science towards society".

References

[1] OMG. Model Driven Architecture. http://www.omg.org/mda/specs.htm. Last accessed on December 10, 2011.

[2] Willian Massami Watanabe, David Fernandes Neto, Thiago Jabur Bittar, and Renata P. M. Fortes. Wcag conformance approach based on model-driven development and webml. In *Proceedings of the 28th ACM International Conference on Design of Communication*, SIGDOC '10, pages 167–174, New York, NY, USA, 2010. ACM.

[3] Danny Groenewegen, Zef Hemel, and Eelco Visser. Separation of concerns and linguistic integration in webdsl. *IEEE Softw.*, 27:31–37, September 2010.

[4] WebDSL. A domain-specific language for developing dynamic web applications with a rich data model. http://webdsl.org. Last accessed on December 12, 2010.

[5] Debasish Ghosh, *DSLs in Action*, Manning Publications, 2010.

[6] Stefano Ceri, Marco Brambilla, and Piero Fraternali. Conceptual modeling: Foundations and applications. chapter The History of WebML Lessons Learned from 10 Years of Model-Driven Development of Web Applications, pages 273–292. Springer-Verlag, Berlin, Heidelberg, 2009.

[7] Guotao Zhuang and Junwei Du. Mda-based modeling and implementation of e-commerce web applications in webml. In *Proceedings of the 2009 Second International Workshop on Computer Science and Engineering - Volume 02*, IWCSE '09, pages 507–510, Washington, DC, USA, 2009, IEEE Computer Society.

[8] Eelco Visser Danny M. Groenewegen. Weaving web applications with webdsl: (demonstration). In *Proceeding of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications*, pages 797–798, 2009.

[9] Danny M. Groenewegen, Zef Hemel, Lennart C.L. Kats, and Eelco Visser. Webdsl: a domain-specific language for dynamic web applications. In *Companion to the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications*, OOPSLA Companion '08, pages 779–780, New York, NY, USA, 2008. ACM.

[10] Z. Hemel, L.C.L Kats, E. Visser *Code Generation by Model Transformation. A Case Study in Transformation Modularity*, Delft University of Technology Software Engineering Research Group 2008.

[11] Eelco Visser *WebDSL: A Case Study in Domain-Specific Language Engineering*, Delft University of Technology Software Engineering Research Group 2008.

[12] Eric Evans, *Domain-Driven Design Quickly*, C4Media Inc 2006.

[13] Nathalie Moreno, Piero Fraternalli, and Antonio Vallecillo. A uml 2.0 profile for webml modeling. In *Workshop proceedings of the sixth international conference on Web engineering*, ICWE '06, New York, NY, USA, 2006, ACM.

[14] Andrea Schauerhuber, Manuel Wimmer, and Elisabeth Kapsammer. Bridging existing web modeling languages to model-driven engineering: a metamodel for webml. In *Workshop proceedings of the sixth international conference on Web engineering*, ICWE '06, New York, NY, USA, 2006, ACM.

[15] K. Czarnecki. *Overview of generative software development. In J.-P. Bantre et al., editors, Unconventional Programming Paradigms (UPP 2004)*, volume 3566 of Lecture Notes in Computer Science, pages 313328, Mont Saint-Michel, France, 2005.

[16] Aldo Bongio Stefano Ceri, Piero Fraternali. *Web Modeling Language (WebML): a modeling language for designing Web sites*, Politecnico di Milano, 2000.

Babeş Bolyai University, Department of Computer Science, 1 M. Kogălniceanu St., 400084 Cluj-Napoca

E-mail address: horatiu@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVII**, Number 1, 2012

A SOFTWARE REPOSITORY AND TOOLSET FOR EMPIRICAL RESEARCH

ARTHUR-JOZSEF MOLNAR

Abstract. This paper proposes a software repository model together with associated tooling and consists of several complex, open-source GUI driven applications ready to be used in empirical software research. We start by providing the rationale for our repository and criteria that guided us in searching for suitable applications. We detail the model of the repository together with associated artifacts and supportive tooling. We detail current applications in the repository together with ways in which it can be further extended. Finally we provide examples of how our repository facilitates research in software visualization and testing.

1. Introduction

Empirical methods are of utmost importance in research, as they allow for the validation of formal models with real-life data. Many recent papers in software research contain sections dedicated to empirical investigation. In [11], Kitchenham et al. propose a set of guidelines for researchers undertaking empirical investigation to help with setting up and analyzing the results of their investigation. Stol et al. survey existing empirical studies concerning open source software in [19], presenting predominant target applications together with employed research methods and directions. More recently, Weyuker overviewed the development of empirical software engineering, highlighting notable success stories and describing current problems in the field [26].

While software literature abounds with empirical research [26], we find a lack of advanced tooling to support such endeavours. Many of the required steps in carrying out empirical research are repetitive in nature: find suitable applications, set them up appropriately, configure them, run the desired experimental procedures and conclude the procedure. Our aim is to assist researchers interested in carrying out empirical investigation by providing an extensible repository of complex open-source software that is already set up

Received by the editors: January 31, 2012.

2010 *Mathematics Subject Classification.* 68N01.

1998 *CR Categories and Descriptors.* D.2.0 [**Software**]: Software Engineering – *General*.

Key words and phrases. Software Repository, Empirical research.

and has multiple associated software artifacts that simplify research in areas such as code analysis, software visualization and testing.

An important step was to study existing research-oriented repositories in order to take necessary steps to address shortcomings of previous approaches. Herraiz et al. [10] describe approaches to obtain research datasets from freely available software repositories such as SourceForge[1] together with associated mining tooling such as FLOSSmole[2] [8]. In [1] Alexander et al. present the *"Software Engineering Research Repository (SERR)"* [1] which consists of source code, software models and testing artifacts that are organized by project. Authors describe SERR as a distributed repository that is easy to extend and which can bring benefits related to software education, research and to provide solid artifact examples for developers.

This paper is structured as follows: the following section details our criteria in searching for suitable applications, while the third section details academic tools used to obtain some of the complex artifacts in our repository. The fourth section is dedicated to detailing our repository's model while the fifth section details its contents. Previous work using the repository is detailed in the sixth section while the last section presents conclusions and future work.

2. Criteria for Suitable Applications

In this section we detail criteria that guided our search for applications suitable as targets for empirical research in domains such as software visualization, analysis and testing. We believe the following criteria to be generally useable in assessing the fitness of candidate software applications as targets of empirical research:

- *Usability*: We believe all meaningful research must target useful software. While a broad term, our opinion is that a vibrant user and developer community that share ideas and actively guide the development of the application to be a good sign of such software. Also, many hosting sites such as SourceForge provide detailed download statistics that can be used to assess real life usage for candidate applications. We present relevant statistics for applications in our repository in the 5th Section of the present paper.

- *Complexity*: Real software is complex and today we have lots of methodologies and metrics for assessing software complexity. Thorough research must be accompanied by several relevant complexity metrics

[1]http://sourceforge.net/
[2]http://flossmole.org/

and should be tied with the usability criteria mentioned above.

- *Authorship*: We believe that picking applications totally unrelated to the research effort goes a long way in proving its general applicability. The best way to achieve this is to select target applications produced by third parties that are uninterested and unrelated to the undertaken research efforts [11].

- *Availability*: The first requirement for using software is of course its availability. This has multiple implications that are best described separately:

  - *Legal*: It is best when target applications are provided under a free software style license. This allows unrestricted access to view or alter source code as needed and the possibility of further distribution within the academic community for purposes of validating research or allowing others access to expand it. In this regard free software licenses[3] compel those who alter the target software to continue providing the modified versions under the same license, preventing legal lock-in.

  - *Technical*: Many research efforts require tracking the evolution of the applications under study. For example, Memon uses a total of 19 software versions of FreeMind [21], GanttProject[4] and JMSN[5] in a case study that attempts to repair GUI test cases for regression testing [16]. In [27], Xie and Memon present a model-based testing approach where a total of 24 versions of four open-source applications are used as case study targets. In addition, our research requires access to multiple target application versions. Our jSET visualization and analysis tool [2] was tested using multiple versions of applications in our repository, which were again reused in our work in heuristically matching equivalent GUI elements [3]. All these efforts were possible by having open-source applications available that have a public source code repository such as SourceForge with available change history going back several years.

[3] http://www.gnu.org/licenses/license-list.html#GPLCompatibleLicenses
[4] http://www.ganttproject.biz
[5] https://sourceforge.net/projects/jmsn

- *Simplicity*: The need to access multiple versions of the same software application raises the importance of the simplicity criterion. In order to have multiple programs set up, configured and ready to run applications should not require complex 3rd party packages such as relational database management systems or web servers that many times are challenging to set up, configure and clean up once investigation has concluded. This has become apparent both in our research [3, 2] and in case studies by Xie and Memon [27, 16], Xiao et al. [18], Do and Rothermel [6, 7] and many others.

3. Related Tools

When developing our repository we made a goal out of harnessing advanced research frameworks to obtain complex artifacts associated with the included applications. In this section we briefly describe the Soot[6] static analysis framework and the GUITAR[7] testing framework which we use to obtain complex artifacts that describe our repository applications.

Soot is a static[8] analysis framework targeting Java bytecode [20, 12, 13]. Currently there are many types of analyses Soot can perform [12, 13], some of which are planned for future integration into our repository. One of the most important artifacts produced by Soot is the application's call graph: a directed graph that describes the calling relations between the target application's methods [12]. The graph's vertices represent methods while the edges model the calling relations between them. Being computed statically, it does not provide information regarding the order methods are called or execution traces. This static callgraph is an over-approximation of all the dynamic callgraphs obtained by running the application over all its possible inputs. Of course, this means the graph will contain spurious edges and nodes, some of which can be eliminated by using mode advanced algorithms [13]. By default, we use the SPARK engine detailed in [12] to obtain callgraphs for applications in our repository.

The second tool we employ is the GUIRipper application part of the comprehensive GUITAR toolset [9]. GUIRipper acts on a GUI driven target application [15] that it runs and records all the widgets' properties across all application windows. It starts the target application and records the properties of all the widgets present on the starting windows and fires events on them[9] with the purpose of opening other application windows that are then

[6] http://www.sable.mcgill.ca/soot

[7] http://sourceforge.net/apps/mediawiki/guitar/index.php?title=GUITAR_Home_Page

[8] It does not run the target application

[9] Clicking buttons, selecting menu items

recorded in turn. The resulting GUI model described in [15] is then persisted in XML format for later use. It is important to note that the only required artifact is the target application in compiled form. Although completely automated, GUIRipper's behaviour can be customized using configuration files. This makes it possible to avoid firing events with unwanted results, such as creating network connections, printing documents and so on. The GUIRipper tool is available in versions that work with Microsoft Windows and Java applications [22]. We currently use the Java implementation of the tool to capture GUI models for applications in our repository.

The following section presents our proposed repository model and details our changes to Soot and GUITAR that allow recording more information about target applications.

4. The Repository Model

Our repository is modeled as a collection of *Projects*, where each *Project* represents a target application at a given moment in time. For example, a project describes the FreeMind application detailed in the 5th Section as found on its SourceForge CVS on November 1st, 2000. Other projects can represent the same application at different points in time. Having multiple projects for the same application helps when studying regression testing, tracking and analyzing software evolution and much more. Each project consists of the following artifacts:

- *Project File.* This is an XML file that contains the project's name and location of associated artifacts. It is similar in function to Eclipse's[10] *.project* and Visual Studio's[11] *.sln* files.
- *Application binaries.* Each project contains two sets of binary files: the compiled application and its required libraries. We currently use different directories for each of these sets of files. Also, the directory containing application binaries contains a script that starts the application. While this may appear trivial, it becomes important when multiple versions of the same application are present. Many times applications record user options and configurations in system-specific locations (e.g: %WINDOWS%/Users) which if not properly handled might cause other versions of the same application to malfunction or

[10] http://eclipse.org/

[11] http://msdn.microsoft.com/en-us/vstudio/aa718325

provide inconsistent behaviour[12]. To mitigate these aspects we manually checked each application version's source code and created adequate startup scripts to set up a consistent environment for the application, which is cleaned after the program's execution.

- *Application sources.* Each project has a source directory that contains the application's source code. The sources should be complete in the sense to allow recompiling the application.
- *GUI Model.* Contains the XML model obtained by running our modified version of GUITAR's GUIRipper on the target application.
- *Widget Screenshots.* Our modified version of the GUIRipper application records screenshots that are associated with the GUI widgets. This allows studying the widgets' appearance without having to start the application. Due to their large number, screenshots are not stored on our SVN repository [23], but are easy to obtain by running the scripts that record the GUI model. This way they will be automatically placed in the correct project subfolder.
- *Application callgraph.* This is the callgraph obtained by running our Soot wrapper over the target application.

Our repository [23] is structured so that every project has its own SVN directory, making it easy to select and download individual projects. In addition to providing actual repository data, our toolset implemented using the Java platform allows programmatic access to target application data. Each project is programmatically represented by an instance of the *jset.project.Project* class which provides access to the project artifacts discussed above. Loading *Project* instances is done via the *jset.project.ProjectService* class that provides the required methods. Our model provides access to method bytecode using the BCEL[13] library that we use to parse compiled code as shown in Figure 3. More so, loaded *jset.project.Project* instances link method bytecode with available sources using Eclipse JDT [24] as a source code parser.

For handling complex artifacts, our repository projects contain all necessary scripts together with Soot and GUIRipper configuration files that allow recomputing the callgraph and GUI model at any time. A lot of work was dispensed when building the required configuration files to make sure all aspects of the target applications are correctly recorded in a repeatable manner. This effort was also reflected in the scripts used when starting the application to make sure that application GUIs remain consistent across multiple executions.

[12]Especially as many applications were not designed to co-exist in multiple versions on the same machine

[13]Bytecode Engineering Library - http://commons.apache.org/bcel

The following sections further detail our model by presenting our changes to the GUITAR and Soot frameworks together with our GUI and callgraph model implementations. Also, we describe how projects can be obtained in an automated build environment.

4.1. **GUI model.** To make the most of available tooling, we started by modifying GUITAR's GUIRipper to provide some additional model information:

- *Widget screenshots.* Every time the GUIRipper records a GUI element it will also save a screenshot of its parent window. The element's associated screenshot together with location and size on the containing window are recorded among its properties so it can be later located by our software tools.
- *Event handlers.* Our modified version of GUIRipper records all event handlers of recorded GUI elements. This provides the link between the GUI layer and its underlying code.

The GUI model is persisted in GUIRipper format, but on programmatically loading the project it is converted into a simpler model we developed to gain independence from external implementations. While adding an extra step, our model provides simplicity while allowing better control when using other model sources such as XAML [14], UIML [17] or HTML. This becomes important as extending our repository beyond the desktop paradigm is a target for future work. Also, by persisting the model in its original format makes it easy to compare models without first having to convert them, which is important when setting up the process for a new target application.

Our hierarchical GUI model is shown in Figure 1. Each GUI is represented using an instance of the *GUI* class, which has no physical correspondent on the actual user interface. Its windows and widgets are represented by *GUIElement* descendants as such: windows are the first level children of the root node while contained widgets descend from their containing windows true to the GUI being modeled. Most GUI element attributes are represented with simple data types such as String or Integer. An important aspect regards the *Id* property which is used by our process to uniquely identify a graphical element. The *Id* must be provided externally to our process as a String instance with the restriction that each element must have a unique identifier. This is enforced in our implementation using safeguard checks that take place when loading the project.

Externally obtained models are converted into our own by providing an implementation for the *IGUIModelTransformer* interface. One such implementation is already available for use with GUIRipper models; other model sources require implementing a suitable transformer.

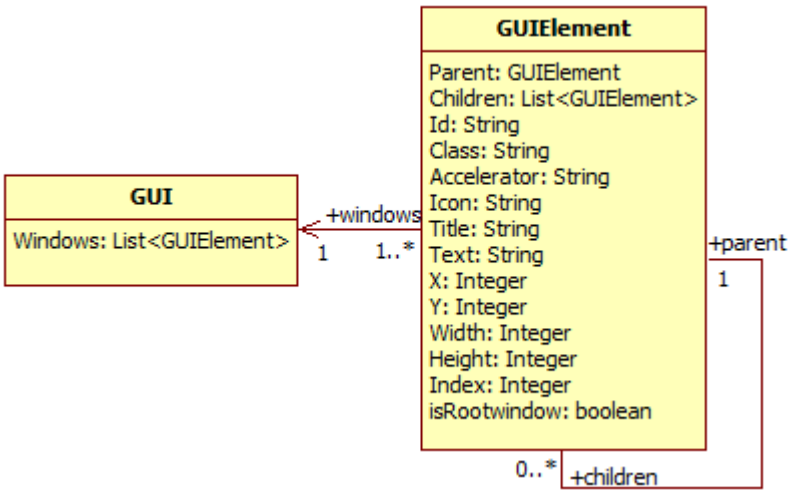


FIGURE 1. GUI model

4.2. **Callgraph Model.** Soot does not provide an endorsed model to persist computed callgraphs so we implemented a wrapper over Soot that persists the callgraph using a JAXB-backed model detailed in Figure 2. The central class is *XmlCallGraph* which holds a collection of *XmlMethod* instances. Note that the callgraph model does not contain classes directly, but they can be inferred using the *inClass* attribute of *XmlMethod* instances.

An important aspect regards the three boolean attributes of the *XmlMethod* class: *inFramework*, *inLibrary* and *inApplication*. When computing the application call graph, we divide analyzed classes into framework, library and application. Framework classes are ones provided by the Java platform itself in libraries such as *rt.jar*, *jce.jar*[14] and so on. Library classes are usually provided on the classpath while application classes are the ones that actually comprise the target software. As the callgraph only models methods and not classes, this information is persisted using them. Also, we must note that multiple classes with the same name might be present in a virtual machine[15], so the three categories are not mutually exclusive.

When a *Project* instance is loaded, the XML model is read and callgraph information is combined with static class data obtained using the BCEL[16]

[14]On the Oracle Java implementation

[15]The Xerces XML library provides such an example

[16]http://commons.apache.org/bcel/

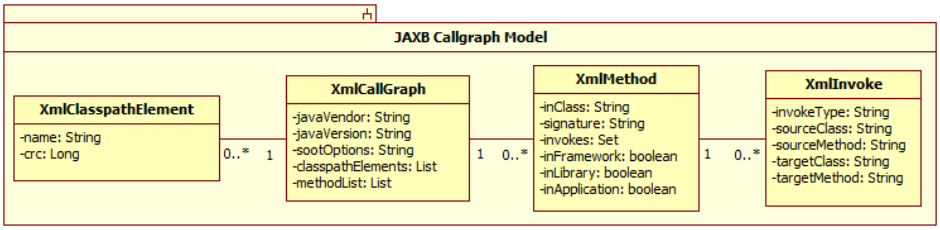


FIGURE 2. JAXB-backed callgraph model

library. The obtained model is shown in Figure 3. *JClass* and *JMethod* instances are wrappers around BCEL implementations and provide class and method level details such as line number tables, constant pools and method instruction lists in human readable form. The model is browsable using methods provided in *JClassRepository*.

Our model's most pressing limitation concerns obtaining the project's code model shown in Figure 3. As BCEL and Soot only work on Java programs, the model cannot be constructed for applications implemented using other platforms. While specialized tools equivalent to BCEL exist for other platforms, we do not have knowledge of tools equivalent to Soot. This leads to the limitation that complete *Project*s can only be recorded for Java software.

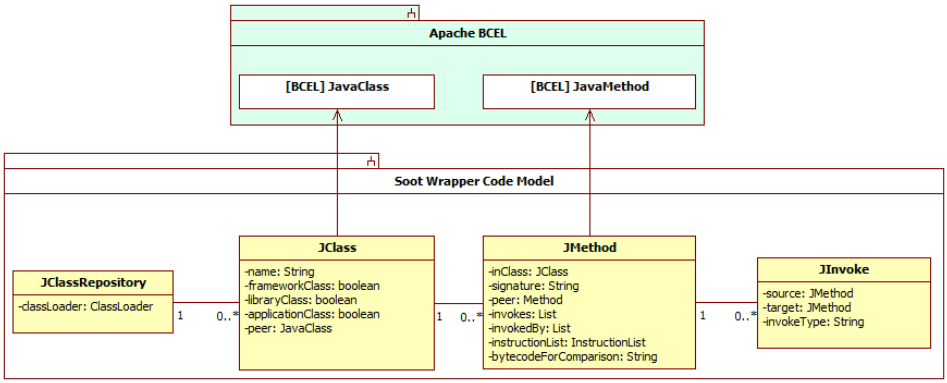


FIGURE 3. Project code model

4.3. **Automatically Building Projects.** Because each project instance captures a snapshot of the target application, it raises the interesting question of

integrating project building with application development. This can help with setting up multiple software versions for research automatically or for purposes of software visualization and analysis [3, 2]. Therefore one of our goals was to provide means to enable automatically building projects from target application sources. Application sources and associated binaries can be obtained using a regular nightly/weekly build process. Obtaining secondary artifacts such as the GUI model and application callgraph is possible using scripts such as ones provided with all the projects currently in our repository.

Automating the process for a new target application requires the creation of suitable script files that compile the application, run our modified version of GUIRipper and Soot wrapper over the compiled sources and place the obtained artifacts into proper directories. Both GUIRipper and our Soot wrapper are extensively customizable using configuration files which only need to be updated in case of major GUI or application classpath changes.

5. Repository Contents

The search based on the criteria laid out in the 2nd section led us to two target applications: the FreeMind [21] mindmapping software and the jEdit [25] text editor. Both of them are available on the SourceForge website and are provided with free-software licenses that allow altering and distributing their source code. The present section discusses both these applications.

5.1. **FreeMind.** The FreeMind mind mapping software is developed on the Java platform and is available under GNU's GPL license on Windows, Linux and Mac platforms. Our repository contains 13 distinct versions of the FreeMind application, dated between November 2000 and September 2007. We used a script to download weekly snapshots of the application starting with the earliest stable version, released as 0.2.0. The development process was not steadily reflected in the source code repository as we found a notable hiatus in CVS data between 2005 and 2006. Out of the weekly versions obtained we found most to be identical regarding source code. Using manual examination we distilled the available data to 13 versions that have differences in source code. Table 1 contains details regarding the downloaded versions such as CVS time, approximate corresponding release version and the number of classes, lines of code and GUI widgets recorded.

Regarding the data shown in Table 1, some clarifications are in order:

- The number of classes[17] includes those generated from XML models, but does not include library or third-party classes.

[17]All the metrics were computed using the Eclipse Metrics plugin http://metrics.sourceforge.net

Version	CVS Timestamp	Classes	LOC	Widgets	Windows
0.1.0	01.11.2000	77	3597	101	1
0.2.0	01.12.2000	90	4101	106	1
0.2.0	01.01.2001	106	4453	132	1
0.3.1	01.04.2001	117	6608	127	1
0.3.1	01.05.2001	121	7255	134	1
0.3.1	01.06.2001	126	7502	136	1
0.3.1	01.07.2001	127	7698	137	1
0.4.0	01.08.2001	127	7708	137	1
0.6.7	01.12.2003	175	11981	244	1
0.6.7	01.01.2004	180	12302	251	1
0.6.7	01.02.2004	182	12619	251	1
0.6.7	01.03.2004	182	12651	251	1
0.8.0	01.09.2007	544	65616	280	1

TABLE 1. Versions of FreeMind used

- The number of lines of code (LOC) includes all non-empty and non-comment lines of code. It therefore includes class and class member declarations and other code that in some cases might be considered as overhead.
- The number of widgets includes all visible or hidden user interface elements recorded by GUIRipper. Therefore this number includes transparent panels employed for grouping controls and elements that might be too small to notice (e.g: 1x1 pixel size) or hidden by Z-ordering.
- Version number is approximative, as sources were downloaded directly from CVS.

With regards to the criteria described in the 2nd section, the FreeMind project was chosen "Project of the Month" in February 2006. The week ending January 29th 2012 saw almost 40.000 downloads, with the total number of downloads being over 14.3 million[18].The current version[19] of the application is 0.9.0 and checking its source control reveals it to be in active development.

By analyzing data in Table 1 certain interesting facts about the application can be noted. First of all we notice the proliferation of Java classes used to build FreeMind: while the first version in our repository only has 77 classes, most later versions have well above 150 classes, topped by the complex 0.8.0 version with over 500 classes. Also, we witness source code line count being in close progression with the number of classes. An interesting aspect regards

[18]http://sourceforge.net/projects/freemind/files/stats/timeline

[19]as of January 30th, 2012

the widget count, that increases by a factor of 2.8 during an 18 fold increase in source line count. Another aspect we must mention is that the only window ripped for FreeMind is its main window. Starting with version 0.6.7, FreeMind also has an *Options* window that could not be correctly ripped and so was left out. While not necessarily a threat to the validity of undertaken research, this aspect must be properly taken into account.

5.2. **jEdit.** The jEdit application is a basic text editor written using Java and similar to FreeMind, is available under the GNU GPL license on multiple platforms. For building our application repository we used a number of 17 versions of this application. While having a public source code repository we chose to select among the many available releases of jEdit for picking the versions. Similar to our approach with FreeMind, we only selected distinct versions (so there are guaranteed code changes between any two versions) that had at least one month of development between them. This allowed us to build a repository containing a reasonable number of applications spread over a number of years with the possibility of including other intermediary versions when required.

It is worth noting that while FreeMind versions were downloaded directly from CVS, for jEdit we used versions that were released by developers and publicly available on the project's download page. This aspect should be taken into account by users of our repository, as it is not unreasonable to assume that FreeMind versions might display more errors or inconsistencies.

The first version of jEdit considered is the 2.3pre2 version available since January 29th, 2000, while the latest version we used is 4.3.2final, released on May 10th, 2010. As with FreeMind, there was a significant hiatus in the development of jEdit between versions 4.2.0final and 4.3.0final respectively. Table 2 presents the versions in our repository together with some key information related to each version, as in the case of the FreeMind application.

Please note that the clarifications detailed in the FreeMind section also apply to the present data, excepting the one referring to software versions.

Studying Table 2 some interesting facts come to light. First of all we can see the recorded metrics changing as jEdit evolves to a more mature version, almost tripling its number of classes and quadrupling the source code line count. Unlike FreeMind, jEdit presents a user interface that consists of more that 10 windows in each of the recorded versions. However while the code metrics increased significantly, the number of windows did not, staying between 12 and 16 across all the versions (although they become more complex by containing an ever increasing number of widgets). Unfortunately, as in the case of FreeMind, jEdit's *Options* window could not be ripped and was therefore excluded from the data above.

Version	CVS Timestamp	Classes	LOC	Widgets	Windows
2.3pre2	29.01.2000	332	23709	482	12
2.3final	11.03.2000	347	25260	533	14
2.4final	23.04.2000	357	25951	559	14
2.5pre5	05.06.2000	416	30949	699	16
2.5final	08.07.2000	418	31085	701	16
2.6pre7	23.09.2000	456	35020	591	12
2.6final	04.11.2000	458	35544	600	12
3.0final	25.12.2000	352	44712	584	13
3.1pre1	10.02.2001	361	45958	590	13
3.1pre3	11.03.2001	361	46165	596	13
3.1final	22.04.2001	373	47136	648	13
3.2final	29.08.2001	430	53735	666	12
4.0final	12.04.2002	504	61918	736	13
4.2pre2	30.05.2003	612	72759	772	13
4.2final	01.12.2004	650	81755	860	14
4.3.0final	23.12.2009	872	106398	992	16
4.3.2final	10.05.2010	872	106510	992	16

TABLE 2. Versions of jEdit used

Regarding the eligibility criteria described in the 2nd section, SourceForge reports over 9.000 downloads during the week ending January 29th 2012 and a total number of over 6.7 million downloads since the project was started at the end of 1999. Also, jEdit was selected as SourceForge "Project of the Month" in October, 2010. Its long development history and available download statistics clearly show that jEdit has a large userbase and is in active development with multiple source code commits in January 2012.

6. Uses of Our Repository

Our repository is already employed in ongoing research regarding topics such as software visualization, program analysis and automated testing. Its first use was to provide target application input data for the software components in our jSET [4, 2] visualization and analysis tool. The multiple application versions were put to good use in our research regarding regression testing of GUI applications [3], an ongoing effort that will greatly benefit from extending our repository. Having repository data at hand freed us from tasks such as setting up target applications and recording the various required artifacts. Also, the large volume of data allowed carrying out detailed studies and drawing compelling conclusions.

Because both applications that form the bulk of our repository data were used in many previous empirical investigations [28, 27, 29, **?**, 30, 5] we believe our repository model and data are relevant for many future research efforts.

7. Conclusions and Future Work

In this paper we presented a proposed software repository model populated with 30 versions of two complex, widely used GUI-driven open-source applications that is freely available for researchers [23]. We detailed changes to popular academic tools that allow recording of key application artifacts together with a software model that allows easy programmatic access to repository *Project* instances. We already employed the presented applications in our research regarding software visualization and analysis [2] and in regression testing GUI applications [3].

As with many repositories, our first and foremost goal lays in extending it. The main limitation of our repository lays in the fact that both included applications are Java-based. The first future direction is to search for and include software based on other platforms such as .NET or Python. Such efforts must be mirrored by finding suitable applications for code analysis that work on the targeted platform and provide similar functionality to BCEL and Soot, allowing us to build suitable application models.

More generally, we must study how to extend our repository model beyond the desktop paradigm so web and mobile applications can be included. Additional research must be carried out on artifacts relevant for other paradigms together with available tools that can be employed to obtain them.

A more distant idea is to switch the repository to a distributed model that simplifies future contribution, thus fueling its growth, like proposed in [1].

Acknowledgements

The author was supported by programs co-financed by The Sectoral Operational Programme Human Resources Development, Contract POS DRU 6/1.5/S/3 - "Doctoral studies: through science towards society"

References

[1] Alexander, R. T., Bieman, J. M., and France, R. B. A software engineering research repository. *SIGSOFT Softw. Eng. Notes 29* (September 2004), 1–4.

[2] Arthur-Jozsef, M. jSET - Java Software Evolution Tracker. In *KEPT-2011 Selected Papers*, Presa Universitara Clujeana, ISSN 2067-1180.

[3] Arthur-Jozsef, M. A heuristic process for GUI widget matching across application versions. *Annales Universitatis. Scientiarum Budapestinensis, Sectio Computatorica* (2012).

[4] **Arthur-Jozsef, M.** jSET - Java Software Evolution Tracker - extended abstract. *KEPT 2011 Conference, Cluj Napoca* (July 2011).

[5] Brooks, P. A., and Memon, A. M. Automated gui testing guided by usage profiles. In *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering* (New York, NY, USA, 2007), ASE '07, ACM, pp. 333–342.

[6] Do, H., and Rothermel, G. An empirical study of regression testing techniques incorporating context and lifetime factors and improved cost-benefit models. In *Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering* (New York, NY, USA, 2006), SIGSOFT '06/FSE-14, ACM, pp. 141–151.

[7] Do, H., and Rothermel, G. Using sensitivity analysis to create simplified economic models for regression testing. In *Proceedings of the 2008 international symposium on Software testing and analysis* (New York, NY, USA, 2008), ISSTA '08, ACM, pp. 51–62.

[8] English, R., and Schweik, C. M. Identifying success and tragedy of floss commons: A preliminary classification of sourceforge.net projects. In *Proceedings of the First International Workshop on Emerging Trends in FLOSS Research and Development* (Washington, DC, USA, 2007), FLOSS '07, IEEE Computer Society, pp. 11–.

[9] Hackner, D., and Memon, A. M. Test case generator for GUITAR. In *ICSE '08: Research Demonstration Track: International Conference on Software Engineering* (Washington, DC, USA, 2008), IEEE Computer Society.

[10] Herraiz, I., Robles, G., and Gonzalez-Barahona, J. M. Research friendly software repositories. In *Proceedings of the joint international and annual ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol) workshops* (New York, NY, USA, 2009), IWPSE-Evol '09, ACM, pp. 19–24.

[11] Kitchenham, B. A., Pfleeger, S. L., Pickard, L. M., Jones, P. W., Hoaglin, D. C., Emam, K. E., and Rosenberg, J. Preliminary guidelines for empirical research in software engineering. *IEEE Trans. Softw. Eng. 28* (August 2002), 721–734.

[12] Lhotak, O. Spark: A flexible point-to analysis framework for java. Tech. rep., McGill University, Montreal, 2002.

[13] Lhotak, O. *Program analysis using binary decision diagrams*. PhD thesis, Montreal, Que., Canada, Canada, 2006. AAINR25195.

[14] MacVittie, L. A. *XAML in a Nutshell (In a Nutshell (O'Reilly))*. O'Reilly Media, Inc., 2006.

[15] Memon, A. M. *A comprehensive framework for testing graphical user interfaces*. PhD thesis, 2001. AAI3026063.

[16] Memon, A. M. Automatically repairing event sequence-based gui test suites for regression testing. *ACM Trans. Softw. Eng. Methodol. 18* (November 2008), 4:1–4:36.

[17] Phanouriou, C. *UIML: A Device-Independent User Interface Markup Language*. PhD thesis, 2000.

[18] Qu, X., Cohen, M. B., and Rothermel, G. Configuration-aware regression testing: an empirical study of sampling and prioritization. In *Proceedings of the 2008 international symposium on Software testing and analysis* (New York, NY, USA, 2008), ISSTA '08, ACM, pp. 75–86.

[19] Stol, K.-J., Babar, M. A., Russo, B., and Fitzgerald, B. The use of empirical methods in open source software research: Facts, trends and future directions. In *Proceedings of the 2009 ICSE Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development* (Washington, DC, USA, 2009), FLOSS '09, IEEE Computer Society, pp. 19–24.

[20] Sundaresan, V. Practical techniques for virtual call resolution in java. Tech. rep., McGill University, 1999.
[21] Website. http://sourceforge.net/projects/freemind/. Home of the FreeMind project.
[22] Website. http://guitar.sourceforge.net/. Home of the GUITAR toolset.
[23] Website. https://sourceforge.net/projects/javaset (Home of our software repository and tooling).
[24] Website. http://www.eclipse.org/jdt (Home of the Eclipse Java development tools).
[25] Website. http://sourceforge.net/projects/jedit/. Home of the jEdit project.
[26] Weyuker, E. J. Empirical software engineering research - the good, the bad, the ugly. In *Proceedings of the 2011 International Symposium on Empirical Software Engineering and Measurement* (Washington, DC, USA, 2011), ESEM '11, IEEE Computer Society, pp. 1–9.
[27] Xie, Q., and Memon, A. M. Model-based testing of community-driven open-source gui applications. In *Proceedings of the 22nd IEEE International Conference on Software Maintenance* (Washington, DC, USA, 2006), IEEE Computer Society, pp. 145–154.
[28] Yuan, X., Cohen, M. B., and Memon, A. M. Gui interaction testing: Incorporating event context, 2011.
[29] Yuan, X., and Memon, A. M. Alternating gui test generation and execution. In *Proceedings of the Testing: Academic & Industrial Conference - Practice and Research Techniques* (Washington, DC, USA, 2008), IEEE Computer Society, pp. 23–32.
[30] Yuan, X., and Memon, A. M. Generating event sequence-based test cases using gui runtime state feedback. *IEEE Transactions on Software Engineering 36* (2010), 81–95.

Department of Computer Science, Faculty of Mathematics and Computer Science, Babeş-Bolyai University, 1, M. Kogalniceanu, Cluj-Napoca 400084, Romania

E-mail address: arthur@cs.ubbcluj.ro

STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LVII**, Number 1, 2012

TOWARDS MDE IMPROVEMENTS FROM INTEGRATED FORMAL VERIFICATIONS

ANNA MEDVE

Abstract. This paper presents a methodology to improve the scenario-based modelling process from model checking and static analyzers during behavioural modelling in MDE process. Our method consists on combining modelling techniques with formal verification techniques to obtain an incremental iterative analysis process in an earlier phase of modelling. This helps not to fix architectural decisions earlier, and to guard and verify some choice for variability. These are based on generic properties of scenario modelling languages and verification tools, presented in the paper.This methodology contributes for future works, to obtain modelling increments from a separate verification engineering service process based in our previous results.

1. Introduction and Motivation

The paper presents a methodology to improve scenarios models by evaluating results of outputs and feedbacks from model checking and static analysis performed with incremental verification process.

As Bezivin stated in 2008 at Dagstuhl seminar [1], for the industrialization of software engineering the individual verification technology (such as model checking, static analysis, or theorem proving) was insufficient and that integrated tool chains and workbenches for model-driven engineering (MDE) were needed. We claim that individual verification workbenches can be very useful for supporting architectural decisions in earlier phases, when the abstraction level is high and deadlock may occurs from incomplete domain attributes. In this later case static analysis combined with model checking can raise the level of genericity in architectural modelling.

Received by the editors: December 10, 2011.

1998 *CR Categories and Descriptors.* D.2 [**SOFTWARE ENGINEERING**]: D.2 Software/Program Verification – *Model checking*; I.6 [**SIMULATION AND MODELING**]: I.6.5 Model Development – *Modeling methodologies*.

Key words and phrases. MDE process, model improvements by verification, scenario-based modelling, formal verification.

We belief that a systematic modelling process for integrating incremental verification should provide model improvements in earlier phases of MDE. For building an MDE process and integrating them with formal verifiers it is necesary to choose verification workbenches and to perform various verification types, or find verification engineering services. As a direct continuation of previous works [2, 15, 16], which introduced verification engineer roles and tasks using Verimag IFx-Omega toolset [12], this work proposes to obtain model improvements based on verification tasks and services.

In this paper we make the following novel contributions towards MDE improvements presented in Section 3 and 4:

(1) We introduce a methodology for integrating incremental formal verification in modelling process. We formulate steps for static analysis and model checking-based scenario validation (in Section 4).
(2) We show concepts and language features from formal scenario-level approach which can be effective to define increments for checking architectural decisions and build a scalable communication architecture (Section 3).
(3) We highlight the effectiveness of our proposition based on research results and trends in scenario-based design.

The rest of paper is structured as follows: Section 2 contains the theoretical background. Section 3 and 4 contains the proposed contribution. Section 5 presents related works and discussion. The closing section contains the few conclusions and future work.

2. Background

2.1. **Model-based Development and Model-driven Engineering.** Model-based development (MDD) is a scalable process which is built on the global model of software and is made up of heterogeneous components [18], which provide several advantages within software paradigms:

- it has techniques on a high level of abstraction;
- it places the model and the consecutive model transformations in the center of the development process;
- it replaces trials with validation and replaces real prototypes with virtual prototypes;
- it has a strong relation among the methods, tools and model management activities during the life cycle;
- it has the basis of support and guidance of design and validation and the basis of the derived low-level descriptions close to implementation.

The advantages and the heterogeneity to be attained require the integration of the means of validation by methodological support in order to increase efficiency, robustness and flexibility in heterogeneous systems.

The Model-Driven Engineering (MDE) standard of OMG specifies that the model represents the system in an expert's observation and the model conforms to its meta model [18]. Regarding the MDE paradigm and our model refinements aims, we supplement the notion of an MDE model and process with conceptual and technical approaches of a given domain.

In MDE process the primary document is the Architecture Model, which is not mandatory Model-Driven Architecture (MDA) if the rapid application development (RAD) is used. The documents of an MDE process are the Requirements Model, Architecture Model and the Implementation Model. The abstraction of the problem lies in the Requirements Model, which we develop into the Implementation Model with the help of systematic model transformations. The application is created from the Implementation Model by code-generation.

The Architecture Model has a central role because it converges with the requirements of the traditional development. This means to describe the architectural decisions in an explicit way in the requirement cluster of the architectural characteristics. The model relationships and roles in MDE have highlighted the architecturally significant requirements, which involve to place the domain-specific knowledges at the basis of architectural decisions.

2.2. **Formal Verification Methods.** Formal verification is a broad term for model checking, static analysis and model based testing of development artifacts formulated with formal methods.

An overview of large family of model checkers is presented at formal method Wiki portal [9, 19].

Model checking is an automated technique that given a finite state model of a system and a property stated in some appropriate logical formalism systematically checks the validity of this property. Model checking is a general approach. Case studies have shown that the incorporation of model checking in the design process does not delay this process more than using simulation and testing and for several case studies the use of model checking has led to shorter development times[6, 10].

Model checking phase consist for checking and reporting. The verification tools operate as a simulator, following one possible execution path through the system and presenting at the model checker GUI the resulting execution trace or a counterexample if faults occur. The methods can be applied to all or only the most critical portions of the system usefully for detecting both basic and other type of errors. The most important is that the procedure is

completely automatic. Typically, the user provides a high level representation of the model and the specification to be checked. The model checking algorithm will either terminate with the answer true, indicating that the model satisfies the specification, or gives a counterexample execution that shows why the formula is not satisfied. The counterexamples are particularly important in finding subtle errors in complex transition systems.

Static analysis

Static code analysis is a general term for a set of techniques used to aid in the verification of computer software without actually executing the programs. The analysis varies greatly depending on the tool employed and analysis guidelines applied in tools [9]. Static analysis is a powerful concept and can significantly aid in development of higher quality software. Specific static analysis tools are style-checking tools, semantic analysis tools, deep-flow static analysis tools, which extend compilation and abstract interpretation capabilities with generic and specific guidelines for checking. A static analysis tool runs automatically and reports all defects it identifies, some of which may be insignificant and require little or no work to correct, whilst others could be critical and need urgent correction. These defects therefore require strong management to ensure that the full benefit is obtained from using the tool in the first place.

2.3. **Attributes for the Selection of Model Checking Tools.** Clarke et al. in [6] argue on orientation toward error detection, as having methods and tools which should support generating counterexamples as a means of debugging and finding errors, rather than for certifying correctness.

We investigated the most important requirements for a verification toolset. A model checking tool contains several input translation engines, the internal or external computation engine for model checking execution, tools to extract the results as outputs and feedback to the input . We enumerate the *desired set of properties for a verification toolset* [20],[10]:

(1) it supports heterogeneity and it integrates model checking with other verification and validation techniques;
(2) it offers combination of features for modelling and validation (i.e. it provides language level access to descriptions and it implements static analysis and optimization techniques);
(3) it is open to modelling languages and validation tools;
(4) it has mechanisms for restricting non-determinism and controlling execution;
(5) it adopts asynchronous execution paradigm to be apt to validate separation of concerns and non-atomic interactions
(6) possibility of QoS predictability;

(7) and last but not least, it is open for all kinds of already implemented components with an adequate interface.

The IFx toolset [12], which we applied in previous work [15, 16] is automaton based and satisfy the above enumerated requirements.

3. Extending MDE Processes by Integrating Incremental Verification

Unlike a complete integrated tool chains and workbenches for MDE, we follow to integrate tools by MDE process which is not intended as a commercial entity. These verification tools are sets of commonly agreed interfaces and operating methods, that allow specific tools to interoperate with other tools forming a complete working environment. This avoids tool dependencies of model-based software applications mainly in earlier phases were multi-language combination became widely used during platform independent modelling.

MDE processes operates dynamically in earlier phases for obtaining the Architecture Model, which is the primary document of a process. The dynamics has strong relationships with requirements model by modelling architecturally significant requirements and architectural decisions. The modelling process involving formal verification is illustrated in Figure 1. The roles for system modelling and for model checking can be performed separately by experts in the fields. The export or import capabilities of modelling tools support safety and efficiency of model transformations needed between tools. This is expressed in XMI/XML standard supporting terms.

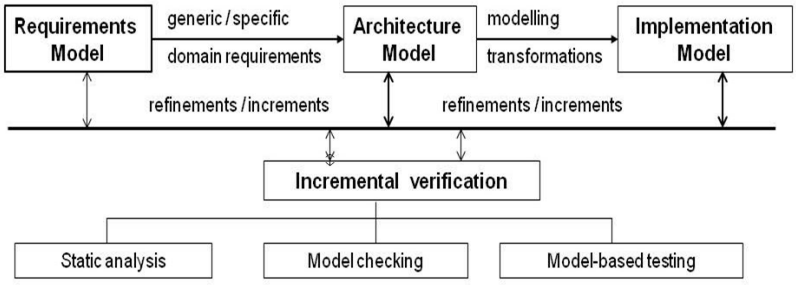


Figure 1. *Main MDE process and documents integrating formal verifications*

Formal verification phase consists of configuring, checking and reporting. The verification tools operate as a simulator, following one possible execution path through the model increment resulting an execution trace and/or a counterexample if faults occur which allows reasoning about model transformations

and to follow incremental verification based on engineering capabilities. Depending on reports content and abstraction level of modelling, combination of automaton based and temporal logic based verification tools may be more fruitful for incremental verification.

Management tasks for integrating incremental verification must involve verification engineering roles and resources, as is reported in [2, 15]. The verification engineering tasks goals are to define and control verification increments in order to discover and eliminate discrepancies between the modelling features. The main process elements are from configuring verification tools for various validation choices; analysis of counterexamples and model checking reports.

3.1. **Explanation of Modelling Process with Integrated Incremental Verification.** We introduce a small example quite complex for showing how a formal scenarios-level approach can be effective to define increments for checking architectural decisions and to build scalable communication architecture. In [14] we introduced the CompConfigur pattern-based method and an example for generic behaviour modelling for the domains which has client-server communication model. This method gives generic domain model in form of a set of class, architecture and scenarios diagrams.

We recall from [14] the example on GMSC (Gateway Mobile Switching Center) call management and we improve it with not covered services. Figure 2 illustrates the model of a simplified GSM network context. The behaviour of a GMSC call management process is restricted to initiate calls received from a local or a remote mobile station (MS, i.e. telephone, intelligent network element), to hande the occurred errors for ensuring the correct performance of connection, and to close the calls.

The static model of GMSC is obtained with CompConfigur architecture pattern language [14] as it shows in Figure 3 and 4. In the case, when the generic architecture pattern not covers every roles and events from a specific domain, it is need to be completed on architectural level for checking their consistency further during the analysis process of detailed design. This case it occurs in recalled example from [14] and it is discussed in this paper, i.e. the architecture model obtained in a way from architecture pattern language is improved by scenario modelling, and model checking.

In the next, by using this example we explain the incremental nature of scenarios and we show the use of scenarios language features, which engage modelling of scenario fragments.

Let the function location of mobile stations that need to be introduced in model. The location service is provided from Basic Station Controllers(BSC)

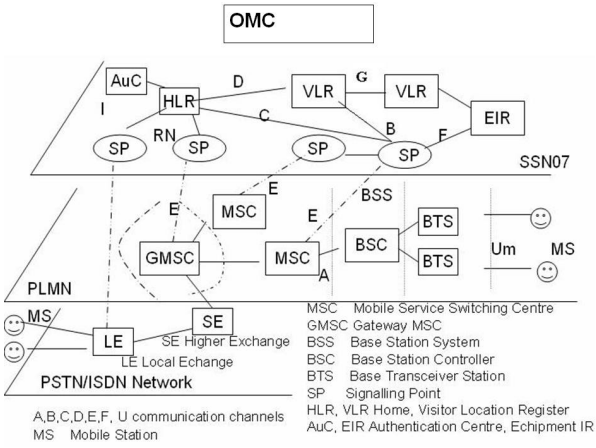


FIGURE 2. *The context of a simplified GSM network*

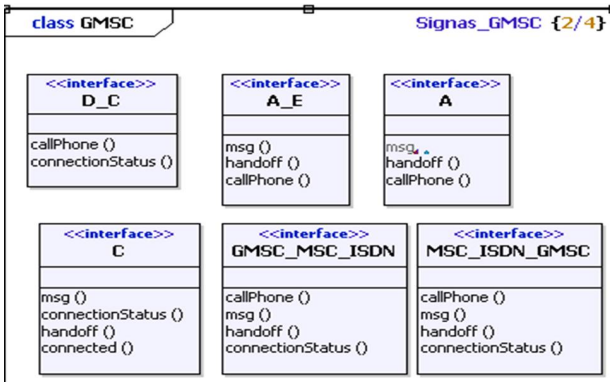


FIGURE 3. *The GMSC class model obtained from applying architectural pattern language*

for Mobile Stations. For this network elements their static model can be obtained from Wrapper Facade architectural pattern. Location service is activated by switching center for user demand. Figure 5 and 6 show two different modelling technique-based scenario models for Mobile Station Location Center (MSLC).

Figure 5 illustrates a bird eye view of the MSL scenario model of the MS location procedure. It serves to follow the message flow to read and find events of MSL function. The composition is difficultly to read, reuse or modify and not supports architectural modelling.

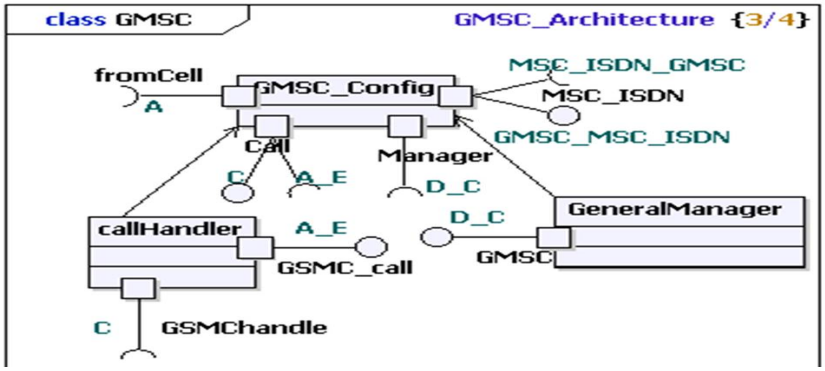


FIGURE 4. *The GMSC architecture model obtained from applying architectural pattern language*

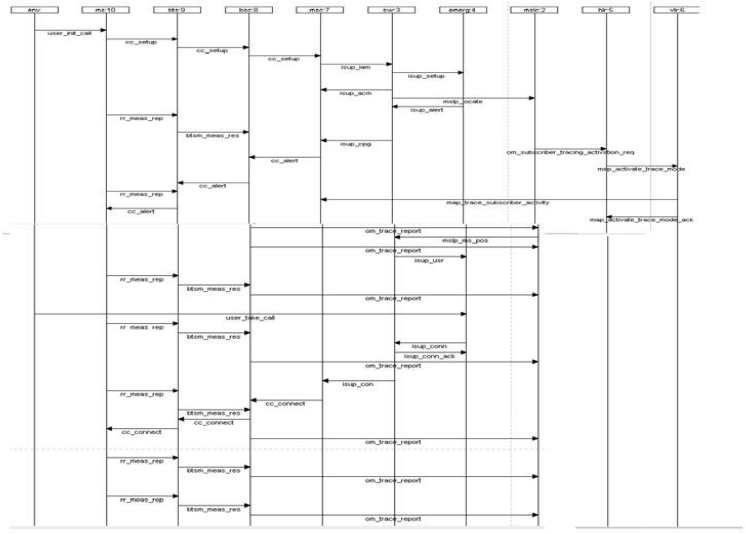


FIGURE 5. *The MSL scenario model in cascade style of message flows.GSM actors from left to rights: ENV, MS, BTS, BSC, SW, EMERG, MSL, HRL,VRL*

Figure 6 shows the MSL scenario model built on referenced subscenarios and messages that prepare the *call* of subscenarios. Referenced scenarios act as remote calls from discovered functionality or event. These form an entity for checking or structuring architectural levels.

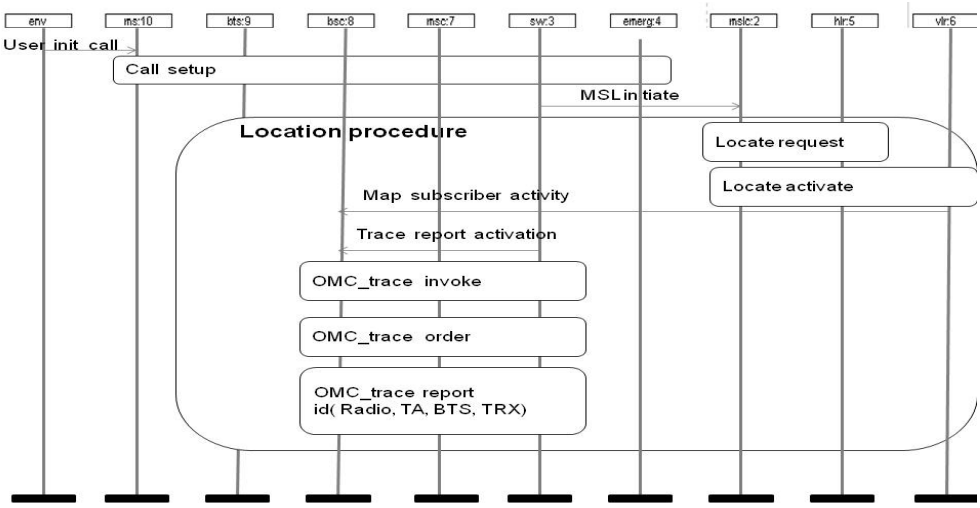


FIGURE 6. *The MSL scenario model in incremental modeling style of scenario fragments and their triggering messages. GSM actors from left to rights: ENV, MS, BTS, BSC, SW, EMERG, MSL, HRL,VRL*

Scenarios fragments allows to follow systematic iterations for incrementing the models and checking for deadline and inconsistencies.

Largely implemented scenario language properties which have results in verification increments are those inter-scenario constraints for searching of pair wise roles in scenarios guided from: scenario reduction from the Inactive Entity; Collaboration-based reduction by reference of one interaction to other interactions within the same (or other) collaboration (remote call); Role decomposition in an interaction into an interaction of its component parts such as *create*; Data composition from observations; - Control- based separations such as *call, kill*; Inline operators for scenario relationships such as *alt, opt, loop, par, corregion, exp, comp, decomp*, as Message Sequence Charts (MSC)[13] language features as well.

Performing model improvements addressing domain properties, such as QoS characteristics, intermediate state transitions to handle error states with fault tolerance for locally corrected, globally nothing bad events, modelling as negative and positive scenarios.

First we perform a scenario-based modelling process to obtain a requirements model from the user views. Next, we augment it with generic and specific domain requirements to obtain an earlier architecture model. This results in

state-based implementation models by integrating in process iteration steps of model checking and model refinements.

The above listed process starts with the modelling interactions between environment and system parts, generic in their nature for the domain. At this modelling phase the major part of the system is abstracted into communication subsystem which assures the safety and performance of the system. To avoid earlier architectural decisions we perform requirements engineering based on generic domain properties and architecturally significant requirements. We can obtain it from architectural pattern language. It results in interaction diagrams for the architecture model.

We introduce an iteration of incremental formal verification. For this an input is needed in the verification workbench. The architecture model resulted from scenario modelling gives the input for verification. Depending on system modelling tools support for XMI transformations we perform a translation process or we translate manually the scenario-based model from earlier architecture modelling.

If the choosed verifier has properties discussed in subsection 2.3 then the input for verifier will be obtained from an XMI or XML translation of the architecture model.

To perform verification engineering an expert modeler or a verification engineering service is needed in process.

In the case of an automaton based model verifier the scenario translation will give the input. In the case of a temporal logic based model verifier the annotated scenario will give a formal basis to construct the input model for the verifier and the input specification as temporal logic expressions.

In the case of manual translation of scenario model into automata's model, which consist of scenario-based model transcription based on conditional annotations for each input signal of a scenario in order to form conditions which behave as states of the emitter-receiver actors in a scenario communication. The receiver will perform a state transition conform to the scenario. It consists of performing one or more actions, decisions, emitting one or more output signals and going in the next state or remaining in the same state.

Some configuration is specified for the static analysis process in order to obtain model transformations or counterexamples. The verification possibilities varies from the abstraction level of the system under verification and the modalities in static analysis followed by deadlock and live lock checking. These are presented in the following section in the form of a set of steps in the verification process and of refinements of the initial system model.

3.2. **Verification Increments for Model Improvements.** *Model improvements based on verification increment:*

The modelling and checking of safety and liveness properties are essentials for absence of deadlock. The deadlock is the general property, where the program stop and makes no further progress.A safety property asserts that nothing bad happens. A liveness property asserts that something good eventually happens. The verification of these properties in earlier modelling phases using small portions of the usage models to obtain increments will improves the quality of architectural modelling.

Static analysis services from verification tools give feedback on unreachable states, unusable variables, or dead code sequences. These result in a refactored model that supports the analysis process in creation of model improvements. The visualization of removed unreachable states and transitions supports observation-control based analysis for fault localization in the design model.

Verification of liveness properties provide useful informations for performing analysis related on architectural decisions, i.e. procedural or functional aspect of the unused variable make explicit the designer biases on functional orthogonality of distributed tasks and variability options resulting in dependency and not of a scalable architecture.

Synthesized model improvements could be obtained from a simultaneously analysis of verification results and corresponding scenario language features localized in scenario model as is indicated in Section 3.1.

In order to obtain granularity of models and to make decisions if the model is complete we observe the results of static analysis for dead variable, i.e the removed variables and signal parameters with their dependency. Setting variations in the configuration file is the way for obtaining the provisioned test by eliminating subsets of functionality.

In a previous experiment in formal verification [15] with a simplified vending machine system we observed significant state space reduction from combination of static analysis tools services before model checking for automated deadlock check.

4. Methodology for integrating incremental formal verification in modelling process

Strategies for configuring verification engineering resources:

First at all, it needs to fix a set of strategy depending on verification engineering goals, services and capabilities, which are considered in the configuration of the behaviour of static analyzer and model checker used:

- for using the automatic configuration in order to not take into account temporal constraints,

- for using a scheduling model in order to describe certain temporal constraints,
- for reducing possibility of state explosion in order to detect possible deadlocks,
- for obtaining the test of the model, which corresponds to an advanced model checking process by configuring the tool features for verifying the model behaviour with the help of temporal logics and other abstractions given in tools,
- for carrying out model increments verification by configuring the static analyzer tool with the behaviour attributes and predefined guidelines from standards and references.

Steps for integrating incremental verification in scenario modelling:

- **Step1: Establish scenarios models as the input to verification.** Create/improve the high-level scenario-based model in MSC or UML2.0 SD. [13]. Export and convert it for input for a given static analysis tool.
- **Step2-2a: Perform static analysis.** Carry all basic features of static analysis (*Live, Reach, Slicing* (the order counts), and *Predefined Guidelines*) and their variations depending on simulation capabilities of the used tool. Iteration can be build from modes of Step8-2b and Step9-2c.
- **Step3: Analyze the reports from verification tool.** Perform an observation-based control to obtain verification increments, which make feedback directly on analyzer configuration and on requirements model.
- **Step4: Return to improve the scenarios.** Decompose static analysis feedback and handle it in order to quantify quality requirements (time, domain, resource), to reduce the number of scenario imprecision; to produce a model consistent domain knowledge and requirements goals; to produce additional information required for development.
- **Step5: Control inter-scenario constraints.** Define and control constraints on interactions for synthesized increments in order to eliminate discrepancies arising in inter-model communication. Inactive entities give support to revision of architectural decisions on task distribution.
- **Step6: Synthesize increments.** The exploration of results from various configuration modes of static analyzer gives verification increments, which improve modelling features as scenario, as use case, as quality requirements, as fault-tolerant scenario category, as input to the next verification act and transformation.

- **Step7: Feedback to new configurations.** Configure configure the verification tool features with abstract descriptions related to time, domain or resource quantifications.
- **Step8-2b: Perform model checking for deadlock check.** Carry out basic features of deadlock checking from verification tool. Iterations will results from combination of Step2-2a and Step9-2c. Follow Step3.
- **Step9-2c: Perform model checking for model-based testing.** Carry all basic features of static analysis (*Live, Reach, Slicing* (the order counts), and *Predefined Guidelines*) and their variations depending on simulation capabilities of the used tool. Iterations will result from combination of Step2-2a and Step8-2b. Follow Step3.
- **Step10-2d: Perform testing.** CADP, SPIN, for carrying out the testing of model, which corresponds to an advanced model checking process by configuring the tool features for verifying the model behaviour with the help of temporal logics and other abstractions given in tools,

Applying those steps the user can observe and validate the specifications in order to be able to decide whether the specification has hiatuses or the model is faulty. This gives feedback for simple safety features which may occur at every run. These may be basic findings starting from: deadlocks, message losses and time settings, or they can be much more specific depending on the domain attributes.

5. Discussion and Related Work

Recent real-world industrial case study on the modelling and validation has stated in Bozga et al. [3] that for such large examples, push-button verification is not sufficient and some iterative combination of analysis and validation is necessary to cope with complexity. They state principles of a verification methodology with three views : systemic view, requirements classification, observation-control based separation of dates.

In [7] Damas et al. present how negative and positive scenarios are the useful tools in requirements engineering for resulting in scenario-based modelling with formal verification which integrate goal, scenario, and state machine models where portions of one model are synthesized from portions of the other models. Our previous works in goal-oriented requirements engineering take account on this win-win engineering framework. These give consistency from iterated goal and improve earlier architecturally decisions.

Werner-Stark et al. in [21] apply systems' theory to obtain granularity and to control the domain quantifiers with qualitative transformations by building

intelligent diagnosis methods for the cases of transient operating conditions, e.g. when it is controlled by an operating procedures.

The Omega UML profile [12, 17] uses IFx to provide a new profile for real-time and embedded systems modelling and verification which extends the expressivity of the currently existing tools. Omega are applied for UML models in other recent works to handle models and model refinements [11]. In future work we apply them to preserve consistency in model improvement steps and to handle the non-determinism of time progress with incremental verification. Visser et al in [22] introduce an iterative technique in which model checking and static analysis are combined to verify large software systems. In their framework the role of the static analysis is to compute partial order information which the model checker uses to reduce the state space, but it not put the focus on earlier architectural decisions and model based verification.

In [5] Frappier et al. describe, utilize at the same example and compare the results from widely used six model checking tools for deciding which of them is the better for the validation of IS specification in model-driven engineering. They do not find generic solutions and not put the focus on verification tools for searching and deciding on the needed verifier capabilities.Their conclusion hold on our hypothesis on the need for MDE process engineering in each development case for optimizing the process among generic and specific domain properties, and existing tools and methods.

Bucchiarone et al in [4] apply in whole process model checking tools as automaton based and temporal logics based, as well for verify and test during entire process of component based web application development. Their method gives generic aspects for formal verification framework building methods. Hannousse et al. in [8] give a generic method for applying static analysis for cross-cutting aspect verification during the composition process.

6. Conclusions and Further Work

We introduced some results of a general approach on integration of formal verification into a scenario-based MDE methodology. Namely, a process and techniques based on static analysis and model checking and scenario-based processes.

In more detail, during the design process one produces a systematically granulated set of scenarios which are used for defining the state-based model of the target system. This model can be then subject to various formal verification tools appropriate for Model Driven Engineering (MDE) (model checking, petri nets, etc.).

Our method for model improvements with incremental verification lies both to formal verification integrated into the tool chains and the workbenches used

by a design methodology, and both to individual verification toolsets. The essence is to draw hope from available verification tools and advances in verification techniques to integrate verification into model-based or model-driven processes.

As future work we plan to apply research results from [7, 17] to work out steps for consistency preservation. Our approach gives rise to a novel conceptual framework for scenario-based model-driven requirements engineering and earlier validation involving model checking. Our results are far from being complete; further analyses and classifications are required based on action research for verification engineering services and tools.

7. Acknowledgements

The authors wish to thank László Kozma who contributed with ideas and help throughout this work.

We thank the anonymous referees for their valuable comments..

This research work was supported by TÁMOP-4.2.1/B-09/1/KMR-2010-0003.

References

[1] J. Bézivin, R.F. Paige, A. Uwe, B. Rumpe, D. Schmidt, : Model Engineering for Complex Systems, Perspectives Workshop: Model Engineering of Complex Systems (MECS), Dagstuhl Seminar Proceedings 08331, http://drops.dagstuhl.de/opus/volltexte/2008/1603.

[2] Zs. Borsi, L. Kozma, A. Medve, One Verification Problems of the Component-based Software Development, 8th International Conference on Applied Informatics, ICAI'2010, Eger, Hungary, 2010, Vol. 2. pp. 391-399.

[3] M. Bozga, S. Graf, L. Mounier, Il. Ober, Iu. Ober, J. Sifakis, The IF toolset, Formal Methods for the Design of Real-Time Systems, LNCS 3185, 2004, pp.237-267, http://www-if.imag.fr/

[4] N. Bucchiarone, H. Muccini, P. Pellicione, P. Pierini, Model-Checking plus Testing: from Software Architecture Analysis to Code Testing, In Proc. Int. Workshop on Integration of Testing Methodologies, FORTE 2004. LNCS 3236 pp. 351-365.

[5] Marc Frappier, Benıt Fraikin, Romain Chossart, Raphael Chane-Yack-Fa, and Mohammed Ouenzar, Comparison of Model Checking Tools for Information Systems,(Eds.: J.S. Dong and H. Zhu , ICFEM 2010, LNCS 6447, pp. 581–596, 2010

[6] E. M. Clarke, E. A. Emerson, J. Sifakis, Model checking: algorithmic verification and debugging, Communications of the ACM Volume 52 , Issue 11, (2009), pp: 74-84.

[7] C.Damas, B.Lambeau, A.van Lamsweerde, Scenarios, Goals, and State Machines: a Win-Win Partnership for Model Synthesis, Proc. FSE'06: Intl. ACM Symposium on the Foundations of Software Engineering, Portland (OR), November 2006., pp. 197-207.

[8] A. Hannousse, R. Douence, G. Ardourel, Static analysis of aspect interaction and composition in component models, 10th ACM International Conference on Generative programming and component engineering, GPCE 2011, pp. 43-52.

[9] formalmethods.wikia.com.

[10] T. Hoare, J. Misra, Verified software: Theories, tools and experiments: Vision of a Grand Challenge project, LNCS 4171 Springer, 2005, pp 1-18.
[11] J.Hooman, H.Kugler, I.Ober, Y.Yushtein, Supporting UML-based Development of Embedded Systems by Formal Techniques, Int. J. of Software and Systems Modeling, Volume 7, Number 2, 2008, pp. 131-155.
[12] IFx-OMEGA Toolset: http://www.irit.fr/ifx/; http://www-if.imag.fr/ OMEGA European Project (IST-33522), http://www-omega.imag.fr/
[13] ITU-T, Message Sequence Charts (MSC), ITU-T Recommendation Z.120, Genova. 1999.
[14] Medve, A., Kozma, L., Ober, I. :General Modelling Approach Based on the Intensive Use of Architectural and Design Patterns, Ed. H. Weghom, P. Isaias, R. Vasiu, Int. Conf.on Applied Computing, IADIS 2010, Timisoara, Romania, October 10-16, 2010, pp. 251-256.
[15] A.Medve, Gy.Orbán, L.Kozma: Let's go verification engineering, 8th Int. Conference on Applied Informatics, Eger, Hungary, January 27-30, 2010, Vol. 2. pp. 417-427.
[16] A. Medve, L. Kozma, MDE process and model improvement using the Verimag IFx verification tools, 8th Joint Conf. on Mathematics and Computer Science, MaCS 2010, Komárno, Slovakia, July 14-17, 2010, Selected p. Published by NOVADAT Ltd. pp. 323-336.
[17] Il.Ober, S. Graf, Iu. Ober, Validating timed UML models by simulation and verification. International Journal of software Tools for Technology Transfer (STTT), Volume 8, Number 2, April, 2006. Springer Verlag, pp 128-145.
[18] OMG , http://www.omg.org/technology/documents/index.htm.
[19] SPIN website: http://spinroot.com/spin/whatispin.html.
[20] VSTTE: Second IFIP Working Conference on Verified Software: Theories, Tools, and Experiments, Oct 6–9, 2008, Toronto, Canada http://qpq.csl.sri.com/vsr/vstte-08.
[21] A.Werner-Stark, E. Németh, K. Hangos, Knowledge-Based Diagnosis of Process Systems Using Procedure HAZID Information, in Knowledge-Based and Intelligent Information and Engineering Systems, LNCS 6883, 2011, pp. 385-394.
[22] W. Visser, G. Brat, Combining Combining Static Analysis and Model Checking for Software Analysis,16th IEEE In. Conf. on Automated software engineering, ASE 2001, IEEE CS, Washington, DC, USA , pp.262-272.

Department of Electrical Engineering and Information Systems, Faculty of Information Technology, University of Pannonia; PhD Student, Faculty of Informatics, Eötvös Loránd University

E-mail address: medve@almos.uni-pannon.hu