MODELLING OF CENTRAL PROCESSING UNIT WORK DENIAL OF SERVICE ATTACKS

Simona Ramanauskaite¹, Antanas Cenys²

¹Siauliai University, Department of Information Technology, Vilniaus st. 141, Siauliai, Lithuania, simram@it.su.lt
²Vilnius Gediminas Technical University, Department of Information Systems, Sauletekio st. 11, Vilnius, Lithuania, ac@fm.vgtu.lt

Abstract. Denial of service attacks is one type of cyber attacks, when attacker tries to make some kind of service not available to its legitimate users. It becomes more relevant for service providers, however, for estimating of the potential threat it is simpler to use mathematical/programmable models instead of the real experiments. DoS attacks can vary in type, as well as its models. In this paper, we propose central processing unit work depletion DoS attack model and its characteristics. Also we use this model to examine attack success probability dependencies on the used queries arrival rate function as well as the optimal queries arrival rate function for the overall attack time period. The proposed model should enrich the other type of DoS models in the way of incorporating them. Meanwhile, the implemented modelling experiments can be used as guidelines for central processing unit work depletion DoS attack efficiency optimization.

Keywords: DoS, Denial of Service, modelling, CPU usage.

1 Introduction

The number of different services in the Internet is increasing every day. Some of these services become more and more popular, sometimes even necessary for some users, functionality of the other systems. Therefore the quality of service becomes an important factor for its proper functioning, not mentioning functioning itself.

Attacks which try to disturb some kind of internet services by denying this service to its legitimate users are called Denial of Service (DoS) attacks. According to CERT [1], in 2010 Denial of Service attacks were in the 6-th place considering how often it occurs in the word. Office of the State Chief Information Security Officer in United States of America ((State of Texas)) stated that DDoS attacks as the way to make ransom attacks will also be one of the most popular attacks in future [5]. Meanwhile, looking into Lithuania in the 3-rd quarter of 2010, this type of attacks was in 4-th place considering occurrence [2].

Usually good preparation can help to avoid DoS attack effect on the certain system and the service it is supplying, however, to estimate the effectiveness of some kind of prevention properties can be quite complex. The aim of this work is to suggest a mathematical model for modelling of central processing unit (CPU) work Denial of Services, and using it, to examine the possible attack success changes when different strategies of the incoming attack flow are used.

2 Denial of Service attacks

Denial of Service attack is a type of cyber attacks. In a denial-of-service (DoS) attack, an attacker attempts to prevent the legitimate users from accessing information or services. By targeting your computer and its network connection or the computers and network of the sites you are trying to use, the attacker may be able to prevent you from accessing email, websites, online accounts (banking, etc.) or other services that rely on the affected computer [11]. In order to increase the attack power, many controlled computers can be used. This kind of attack is also known as Distributed Denial of Service (DDoS) attack.

Practical experiments with DoS and DDoS attacks are difficult because of these reasons:

- The attack sources area spreads in the wide geographical area, and experiments in the local network can be insufficient to illustrate the real situation;
- DDoS attacks require a lot of controlled computers and can cause a difficulties getting the sufficient amount of infected and ready to attack computers;
- Real experiments in the Internet can cause problems for third parties, disturb the work of innocent Internet users or even services;
- Execution on DoS attack in the Internet can be illegal.

Examination of the attack properties without the real execution of DoS attacks can be done using different modelling methods and tools. However talking about modelling of DoS attacks, there can be different types of DoS attacks, so different models must be used to model the desired attack situation.
There are three main categories in DoS attack classification concerning resource, which can be exhausted to cause the DoS effect:

- Bandwidth exhaustion DoS attacks;
- Memory depletion DoS attacks;
- Central processing unit work depletion DoS attacks.

2.1 Bandwidth exhaustion DoS attack

Bandwidth exhaustion DoS attack happens when an intruder consumes all the available bandwidth on a certain network by generating a large number of packets directed to your network. Typically, these packets are ICMP ECHO packets but in principle they may be anything [8].

Qiang Huang, Hisashi Kobayashi and Bede Liu suggested two models for modelling bandwidth exhaustion DoS attacks. One of them is for DoS attacks in the global network [9], and the other one is for the wireless networks [10]. These two models offer the methods allowing finding the minimal number of agents necessary to execute the successful DDoS attacks, however, the models do not pay enough attention to the properties of all attacks. More attack properties are taken into account in the paper focusing on the modelling of DoS attacks using stochastic methods [14]. Also this paper represents mathematical expression for quantitative DoS attack success calculation using the known data on the attacks, normal flow and other victim’s properties.

2.2 Memory depletion DoS attack

DoS resource depletion attacks involve the attacker sending packets that misuse network protocol communications or sending malformed packets that tie up network resources so that none are left for legitimate users [15].

Memory depletion DoS attacks are the most common because of the noticeable effect and quite low attack expenses. This is why there is a quite big range of proposed memory depletion DoS attacks models:

- Q. Huang and other authors of the paper “Analysis of a New Form of Distributed Denial of Service Attack” [9] apply the simplified Engest loss model G(N)/G/m(0). This model enables to estimate the success of SYN flooding attack but there are no characteristics of the legitimate users in this model, only the attack itself is characterised;
- The author of “Defending against Flooding-Based Distributed Denial-of-Service Attacks: A Tutorial” [3], uses G/D/∞/N model to calculate the minimal attack flow, which is necessary to make a successful TCP SYN attack. However, in this work the model is not described in detail, and only the results of the experiment are given.
- Y. Wang and the other authors use the two-dimensional embedded Markov chain model in the paper “A queuing analysis for the denial of service (DoS) attacks in computer networks” [17]. This model takes into account legitimate and attack flow characteristics as well as the buffer size. But this model is difficult to use because of the complex calculations.
- Authors of the paper „Modelling of SYN Flooding Attacks“ [13] suggest the SYN flooding attack model which can be used for any kind of the memory depletion DoS attacks and allows to estimate the success probability of the attack, taking into account both victim and attacker properties.

2.3 CPU work depletion DoS attacks

CPU work depletion DoS attacks differ from memory depletion attacks considering the exhausted recourse type. This type of attacks attempts to make victim to do some kind of job which pretends to seem more important than the legitimate user requests or requires more time to finish it. For these reasons, the legitimate user requests are served slower and even the impression of not execution at all may appear as the attack requests use up CPU processing cycles to prevent the legitimate processes from being processed.

This is one of the oldest known forms of denial of service and mature operating systems most often have the defence, nevertheless, some of them are still vulnerable to this type of DoS attacks [6].

Maybe because of variety of different systems and varying request service strategies in them, CPU work depletion DoS attacks do not have a huge effect on it. It is the same as with the models of this type of DoS attacks. There exist different models for CPU performance and scheduling algorithm analysis [4, 7, 12], however the impact to CPU work during DoS are not analyzed widely. This is why we suggest a CPU work depletion DoS attack mathematical model, which would let judge the success of this type of DoS attacks with certain attacker and victim properties configurations.
2.3.1 Conceptual model of CPU work depletion DoS attack

The basic idea of CPU work depletion DoS attack model should be that there exist the criteria allowing judging what might be the success of this type of attack. It would be wrong to judge the success of attack considering the processor utilization percentage because the legitimate user does not care about the system itself but just about the service he gets. This means the success of attack should be judged considering the time necessary to process the legitimate user query.

Thinking about CPU work, it is also important to remember the fact that one CPU can serve just one process at one time moment and the illusion of multiprocessing is achieved by scheduling small parts of certain tasks. In this work, only those situations are analyzed where the non-priority scheduling algorithms are used and where the legitimate and attack requests are not distinguished. Also we assume that CPU work depletion Does attacks do not block incoming requests, and the data buffer to store requests is infinitive. This is because the control of incoming requests blocking is done by memory – not CPU work – management systems.

To fully define the CPU work model we use these properties:

- CPU takes one query and process it for a \( k \) cycles within one take;
- Legitimate queries require approximately \( c_l \) takes in to CPU;
- Attacks queries require approximately \( c_a \) takes in to CPU;
- At one time moment there are \( M \) queries in the buffer;
- There are \( p_l \) percent of legitimate and \( 1-p_l \) percent of attack queries in the system;
- The processor frequency is \( f \), which means how many cycles can be processed by the processor in one time moment (second).

We assume that legitimate and attack flows are distributed evenly. This allows estimation of the average number of takes into CPU for one query (how many times query should be processed by CPU to finish it):

\[
C = c_l \cdot p_l + c_a \cdot p_a
\]

(1)

For deeper analysis of query processing in CPU, we should know what scheduling algorithm is used. One of the most popular is FIFO (First In – First Out). Using this scheduling algorithm queries are not divided into small blocks and they all are processed instantly. Immediately after the processing of one query, the other is processed – then CPU finishes processing of the query, and the query is removed from the buffer.

![Figure 1. Conceptual model for serving of FIFO query](image1)

To estimate the average time of certain query, we should not forget that incoming query goes to the end of the queue (buffer), so before this query will be processed in CPU, all the previous queries must be processed either.

\[
T = \frac{c \cdot (M \cdot k)}{f}
\]

(2)

Another popular non-priority scheduling algorithm is “Round-robin”. Using this scheduling algorithm, all the queries are processed in small parts. When one small part of query is executed – and if it was not the last part of query – this query “goes” back to the end of the queue (buffer). And just after executing the last part of it, the query is removed from the queue (buffer).

![Figure 2. Conceptual model for serving of Round-robin query](image2)
To estimate the average time of certain query, we should count how many queries are in front of the watched one every time the query goes back to the end of queue (buffer). So we have to express the size of buffer as a time function \( M(i)=M(i-1)-M_{\text{done}}(i-1)+M_{\text{new}}(i-1) \), which would be estimated by queue size one moment ago, adding newly coming queries and subtracting number of queries finished in the previous time moment. So the average service time could be expressed as following:

\[
T = \sum_{i=1}^{f} c \cdot M(i)
\]  

Using this formula the average query service time can be estimated quite precisely even when size of queue changes dynamically. But usually the buffer size is quite constant, for example, in those situations, when attacker sends so many queries that the buffer is fully filled and even some of them are blocked (especially in DDoS attacks). In such situations the average service time formula can be simplified and would be the same as in FIFO scheduling algorithm.

2.3.2 Estimation of attack success probability in CPU work depletion DoS attack model

No matter FIFO or “Round-robin” scheduling algorithm is used, to estimate the attack probability there must be known the critical request service time limit \( T_{\text{max}} \). It shows how long the legitimate user is really to wait until it is assumed that the request did not reach the service or was rejected. So the attack success probability can be expressed as ratio of average and critical service times. And in these situations when the average service time is greater than the critical service time, the attack success probability is equal to 100%:

\[
P = \begin{cases} 
\frac{T}{T_{\text{max}}} & , T \leq T_{\text{max}} \\
1 & , T > T_{\text{max}} 
\end{cases}
\]  

2.3.3 Modelling experiments

The proposed CPU work depletion DoS attacks model formulas and its modelling result show the dependencies between the attack and victim properties. The average attacks success probability linearly depends on the average query service time (number of cycles necessary to finish legitimate and attack query and its quantity distribution in the queue) and queue size in the buffer and is inverse to CPU frequency and critical service time. Similar dependencies can be noticed in experiments of the other authors [16, figure 11].

![Figure 3. Attack success probability changes in buffer size increase, depending on used cycles to execute one attack query](image1)

![Figure 4. Average query execution time changes in CPU frequency, depending on used cycles to execute one attack query](image2)

To estimate the attack success changes using different incoming queries generation strategies, different incoming queries arrival functions \( v(t) \) were chosen, and all of them for the observed time provided the same number of queries to serve:

- Constant arrival rate:
  - Arrival rate is less than serving rate – \( v(t)=10 \text{ qps} \);
  - Arrival rate is equal to the serving rate – \( v(t)=1/T \text{ qps} \);
  - Arrival rate is greater than serving rate – \( v(t)=10000 \text{ qps} \).

- Linearly changing arrival rate:
  - Arrival rate is increasing in time – \( v(t)=20t \text{ qps} \);
  - Arrival rate is decreasing in time – \( v(t)=10000 - 100t \text{ qps} \).
Using these query arrival functions with the proposed CPU work depletion DoS model, we noticed such tendencies:

- When arrival rate is the same as service, attack success probability stays the same all the time;
- When arrival rate is less then service time, the success probability decreases (but slower), and then it increases with higher arrival rate then the service rate;
- When arrival rate is constant, the attack success probability changes linearly;
- When arrival rate changes in time linearly, the success probability increases exponentially;
- When attacker starts with high arrival rate and decreases it linearly in time, the success probability is increasing until the rate reaches 0 value and then start to decrease.

Figure 5. Attack success probability changes in time, depending on queries arrival function

In addition to the examination of the success probability curve changes in time depending to queries arrival rate, the analysis of the arrival rate efficiency considering attack success was done as well. These tests showed that the most efficient are the arrival rate functions when the number of queries in the buffer is big at the start point even if latter it can decrease. Meanwhile, if attack would use a linearly increasing arrival rate, the attack success would change more rapidly, however, it would be more sufficient only at the end of attack, and not during the overall attack time.

3 Conclusions

1. There are many different models for memory depletion and few for bandwidth exhaustion DoS attacks, however, practically none for CPU work depletion DoS attacks. This may be because of not popularity of this type of DoS attacks and quite little success probability; nevertheless, it is necessary to make deeper analysis of the different type of DoS attacks and more precise value estimation of the other types of DoS attack properties.

2. CPU work depletion DoS attack model is assumed considering that the buffer size is infinitive. This means this model should not be used separately for real situation modelling, and it should be combined with memory depletion DoS attack model. Such combination would work more realistically for both – CPU work and memory depletion DoS attack – models.

3. The average execution period for user queries in CPU work depletion DoS attacks depends not only on time necessary to execute the query but on the queue size as well. So if the incoming traffic increased adequately considering both parameters – legitimate and attack – the success probability would increase linearly as well because of the increasing queue size. But if attack traffic increased and the legitimate traffic remained constant, the success probability should increase not linearly, but exponentially.

4. Attack success probability in time changes accordingly to the arriving query rate function. To get the most effect considering the overall attack time, the instant big query arriving rate should be used, and what can be achieved choosing a certain query arriving rate or combining some of them in time is the desired attack effectiveness.
References


DISTRIBUTED SYSTEM MODEL CHECKING DESIGN

Robertas Jasaitis, Eduardas Bareisa

Kaunas University of Technology, Software Engineering Department,
Studentu St. 50, Kaunas, Lithuania, jasaitis.robertas@gmail.com, edas@soften.ktu.lt

Abstract. Testing is a very important but also very expensive stage of software development. Most applications today communicate with other components over the network. The testing of such application is often very complicated due to the interleaving of the clients and servers. To make testing of distributed systems easier we propose an algorithm for networked application source code model checking which could be a part of such systems testing. We simulate two client application instances at one time and try out different communication with server application execution scenarios. It’s obvious that client application should communicate with the server at least twice, otherwise there is no need for such testing. In this paper we report a first verification attempt based on proposed algorithm with a small sample application build on the J2EE framework.

Keywords: Distributed system, model checking, testing.

1 Introduction

More and more applications today are designed to be networked, because of the rapid spread and power of internet. Some components are often moved to some servers and so can be reused by client applications [7, 8]. Servers can provide good performance because of the hardware which is very important for growing popularity of mobile devices. Network applications are multithreaded by nature so model checking is very important for verification of such applications [2].

Networked applications often face issues related to several client instances racing each other while communicating with the same server, especially when sending similar requests to the server [3]. Even if an application is single threaded there is no guarantee that it will not be affected by other instance of the application communicating with the same server at the same time. Testing this situation is complicated because of two main problems:

1. Two instances of the client application (system under test, SUT) run on different processes so they are not under control of the same model checker [1].
2. Even if we could model check the two instances of SUT we could not backtrack the communication operations if they are not atomic, which is the case in most server implementations.

Our work proposes solution to the first problem described above and the second problem is now workaround as our focus is to solve problem 1 at the moment. We provide simple reservation application example which helps to understand the problem and the solution.

This paper is organized as follows: the intuition and example of our solution are given in section 2. Implementation of the algorithm is given in section 3. Limitations of our algorithm are given in section 4. Experiments are given in section 5. Conclusions and future work are given in section 6.

2 Intuition of the Algorithm

We start this section with an example in order to get better understanding of the problem and get better understanding of the proposed solution. We analyze “restaurant table reservation system”. This system is built on J2EE framework [6]. The usual work flow of such application is described by the following sequence:

- User starts client application
- User checks which tables are available
- User selects one.

The simplified application implementation is shown below:

```java
public static void main(String[] args) {
    int tableId = 3;
    if (isTableAvailable(tableId)) {
        reserveTable(tableId);
    }
}
```

We do not go deeper into isTableAvailable and reserveTable methods for simplicity. The first “isTableAvailable” method opens URL connection, sends request to server and waits for the response just to check whether the given table is not reserved yet. There is a http servlet implemented and running on the server which handles the requests. The second “reserveTable” method sends request to server to mark the given table as
reserved. In this example we hardcoded the table number as selection of the data is not the target of the
algorithm.

The code is very clear and obviously correct if there is only one user (only one application instance)
working at a time. What would happen if there would be two instances of the application running at the same
time? That’s what our algorithm tries to analyze and answer for us. Our algorithm finds the connection code in
the application and simulates two application instances and tries different connection code sequences. Figure 1
shows all possible sequences of our sample application. In the given situation there are two instances of the
application running at the same time. We treat that the “isTableAvailable” and “reserveTable” methods are
atomic actions. In the figure these actions are marked by “check” and “reserve” keywords respectively. S1 - S6
marks the indexes of the sequences. So in this case there might be 6 possible sequences.

Figure 1. All possible sequences of restaurant table reservation application

The graph links stand for actions (one of our two actions: check availability and reserve table) and the
nodes stand for model checker decision points - the decision with action to choose from the available actions at
the moment in figure 1. So as mentioned earlier we have 6 possible action sequences (paths). The sequences S1
and S4 are shorter than other sequences. This happens because of our sample code: we first check if given table
is available from first client application then reserve it from the same instance of application. Then we check
whether the table is available from the other instance of application. So the server now returns that the table is
not available so we do not get into the “if” clause in our sample code, so the second reserve table action is never
called in those sequences. These two sequences fail to detect any issue in the client/server implementation so we
could ignore those in our example. From the sequences S2, S3, S5 and S6 we can see that the very same table
has been reserved twice: have been reserved for both client application instances which is an issue in the code.
This obvious issue could be resolved on a database level, or file system, or application server but in a real life
system this algorithm could help detect thread “unsafe” environment which could be a result of bad design, or
simple because of faulty coding.

When algorithm does the model checking it uses code backtracking [4]. Let’s take a look at figure 1 one
more time. The algorithm would start with the sequence S1 in our sample situation. So it would go from node 1
to node 4 and then would backtrack to node 3 and then to node 2 as there is no other possible path from node 3.
So as we now are in node 2 the algorithm would go to node 5 which means that the request would be sent to the
server, but the response would be different then we expect in the model as the table is already reserved. This happens because when the algorithm does backtracking the server is not backtracked [2]. To solve this drawback we now introduce the requirement to implement “reset” request on the server. When this request is received the server should restore its state to the initial testing mode state. The actions till the node 2 (in this case it is only one “check” action) are repeated before going to node 5. As we mentioned in the introduction, this issue is not the focus of the article so this is just a workaround of this problem which let’s our algorithm work correctly. We are planning to solve this problem properly later.

3 Implementation of the algorithm

Our algorithm is a networked application model checker, so its implementation can be based on some existing single instance application model checker. We have chosen to use one of the most popular Java model checkers - Java Path Finder (JPF) [5]. JPF is very handy for us as it’s an explicit-state model checker for Java programs and is build on top of a custom-made Java Virtual Machine [5].

JPF uses symbolic execution so can reach any code block in the application [4]. JPF is also able to generate required test inputs [10], so we do not have to worry about this. Our implementation is based on top of the JPF. Our JPF extended version (we call it EJPF) is working the same as the original one until it reaches the code which opens http connection. At this point the EJPF “remembers” this position and continues application execution till the application is finished. During this execution EJPF collects the information about other http connections found in the application. After this EJPF backtracks to first “remembered” position and executes the code again, imitating the new instance of application. So after this the first sequence has been executed and all the information about the http connections has been collected. Now we can back track and test different sequences according to the remembered http connections. In such a way we imitate two application instances running at the same time.

Each time JPF does backtracking and chooses new path we need to send out the “reset” request to the server and repeat the actions till our current point. As we mentioned earlier this is just a workaround as this problem is not the focus of this article.

4 Limitation of the algorithm

The main limitation of this algorithm currently is the requirement to have the “reset” request feature on the server. We are now working on this problem and so this is both: the limitation of the algorithm and our future work. By having this solved properly our algorithm would even become simpler. In such case we would not have to change the server in any way so this algorithm would be more convenient for the users so this task is very important for us.

5 Experiments

This algorithm is not fully implemented yet so we were able to do some minor experiments only. To check the algorithm correctness we check how many different sequences are generated by the algorithm [9]. We have chosen two simple applications. The first one is the restaurant table reservation application we analyzed earlier which does two requests. The second one is cinema theater seat reservation application. This application sends three requests: gets all available movies that are shown today, checks all available seats of the movie, reserves the seat. The results are displayed in the table 1.

<table>
<thead>
<tr>
<th>Application</th>
<th>Requests count</th>
<th>Sequences generated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restaurant table reservation</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>Cinema reservation</td>
<td>3</td>
<td>20</td>
</tr>
</tbody>
</table>

We have chosen small applications so that we could check all the sequences are generated correctly. We can see that only 3 requests generates 20 sequences so it would be hard to check the correctness of the algorithm using application with big number of requests.

6 Conclusions and Future Work

Networked applications often faces issues related to several client instances racing each other while communicating with the same server, especially when sending similar requests to the server. The file becomes even more complicated if several servers exists in the distributed system. Our proposed algorithm helps detect this type of problems without making changes to the application under test.

The given algorithm doesn’t find issues in the code so doesn’t solve oracle problem, it’s a networked application model checker rather than tester. So this technique cannot be used as a standalone tool for testing so it still requires additional testing techniques like Unit tests or some other. Our model checker helps to find
problems related to multiple instances of application racing each other while communicating with the same server or servers.

The algorithm still requires several things that should be done in the future:

1. We need to find out some solution for the server request backtracking. In this solution we used workaround so this have to be implemented properly.
2. Research some way to extend this model checker algorithm so that it could detect problems and report about weak or risky code blocks.

References


OPTIMIZATION OF E-SERVICE DEVELOPMENT FOR LATVIAN PUBLIC SECTOR

Pavel Semenchuk
Riga Technical University, Applied Computer Science Department,
Meza str. 1/3, Riga, Latvia, Pavels.Semenucks@rtu.lv

Abstract. The development of Latvian State E-service was started in January, 2006 (Integrated State Information System (ISIS, Latvian abbreviation IVIS)). The customer and provider of the system is Secretariat of Special Assignments Minister for Electronic Government Affairs (SSAMEGA)). Since the middle of 2007 the infrastructure is accessible for E-service designers and developers. ISIS is based on Service Oriented Architecture (SOA) principles and includes many individual innovations and modern solutions. There are more than 60 E-services, which were developed using ISIS infrastructure, already accessible for Latvian inhabitants on Latvian government E-service portal (www.latvija.lv). These E-services consist of many Web services which are developed by different organizations and published in special register. The number of E-services growths up very quick. To avoid duplication of Web services from different organizations new solution was worked out and soon will be implemented in ISIS. This and other ISIS innovations are described further in this paper.

Keywords: e-government, SOA, logic programming, deductive database.

1 Introduction
Development of technologies provides new and effective methods all the time which allow improving work of data procession as well as mutual exchange and integration. These are important processes which have to be considered in planning and development of State Electronic Administration because in work of state administration are used several substantial registers where data about objects and subjects existing in their contents are stored as well as about services which the state has to provide according to legislation. Electronic service development and state electronic government implementation is one of priorities almost in all the most developed countries of the European Union and the world, Latvia included. It is very important that e-government and integration process would occur not chaotically, but unitarily. Preceding decade could be considered as time, when primary data registers were created. The new decade, evidently, will be oriented on these registers integration, offered services electronization and their accessibility for inhabitants. All developed countries of the world are going this way now.

Before ISIS system creation the world and another Baltic state experience was taken into account. Major part of attention was fixed on Estonian X-Road [12] system. ISIS was created on X-Road successfully usage principle: defined centralized data exchange point, standardized data exchange formats and protocols. The main difference is that ISIS tries to use already existing state registers and information systems data exchange infrastructure, if it exists. At the moment X-Road integrates 150 data bases and information systems and is used widely, but ISIS usage at the moment is only for e-services functionality providing. Lithuanian integration project (e-Gate) [4] goes similarly as ISIS project. It was started in 2004. In 2008 is planned, that two pilot e-services will be created for e-Gate project.

The article’s author have participated in development (system analysis, design, coding, and integration) of ISIS system from the very beginning, and ensured successful defence of the system publically in Riga Technical University State Qualification Commission. It proceeds to be developed during development process of Doctorate work. By use of ISIS infrastructure were developed already more than 60 E-services for Latvian inhabitants.

This paper is organized as follows. Section 2 describes the ISIS architecture. Section 3 describes the principals of logic programming and deductive data bases. Section 4 describes the optimization solutions for ISIS architecture and E-service development. Conclusions state further directions of research.

2 Description of ISIS solution
ISIS ensures a possibility mutually integrate state and municipal information systems, which now or in the future will be providers or recipients of various E-services including numerous portals, from which E-services will be accessible for inhabitants. Not only state or municipal institutions will take part in the development of the E-services, but also commercial institutions, for example, banks, which will be necessary to involve for paid E-services realization.
ISIS solution is based on the following principles:

- The use of Service Oriented Architecture (SOA).
- The solution technologically supports the integration with numerous E-service providers.
- Identification standard, which provides world scale unique number assignment.
- The use of state level unified standards for E-service and integration projects development - XML schemas development standards [11], IS service development standards [5], E-service development standards [3], SOA standards.
- Four centralized catalogues development – XML schemas catalogue, IS service catalogue, E-service catalogue, and public services catalogue.
- Common use E-services and IS services are technologically accessible from many access points – state and municipal portals, institution home pages, etc.
- Use of E-service envelope, when implementing asynchronous E-service.
- Safe mailboxes providing for office workers and inhabitants with possibility perform tasks, which are provided by E-service during implementation process.

ISIS solution conceptual description is shown on the Figure 1. Registers are related to integration software. For each data object, which is required for E-service realization, is necessary to work out XML schemas set.

This process is regulated by the corresponding standards and guidelines and is used for XML schema basic objects and types description. Data call from the corresponding register is performed by Web services help (in ISIS context these are named as IS services). During IS services calls occurs metadata sending, which describes the call. With the metadata register provider will receive the information, which is necessary for him to fill in his system’s audit notes about a specific request. IS service development and using also is regulated by a corresponding standard. IS services are divided into two groups: Business IS services (includes some specific business activities, for example, get inhabitant’s first name and last name by the personal code) and Integration IS services (includes some other Business IS service calls and E-service business logic). Implementation of Integration IS services usually is provided by the BizTalk server – one of the basic E-service infrastructure parts.
ISIS solution architecture comprises the following base elements:

- hierarchy of XML structures;
- IS service;
- e-service.

Platform for e-service implementation are made by several components which are:

- external portals and application which are made by:
  - e-service presentation layer;
  - external services and business logic libraries.
- ISIS infrastructure which is made by:
  - request service;
  - common use services of ISIS infrastructure;
  - integration IS services;
  - orchestration IS services (IS services made on BizTalk base what ensures operation of asynchronous and multi-step e-services);
  - business IS services.
- Providers of authentication and authorization information: management module of ISIS users and groups of users (MMUGU), inhabitants’ accounts of state portal, safe electronic signature of JSC Latvia Post;
- ISIS applications;
- IS of state and municipalities.

Conceptual description of ISIS solution is available [8].

2.1 IS service catalogue

IS service catalogue is a part of ISIS architecture. It forms the unified repository for ISIS IS services which are used for E-service creation. In ISIS architecture with IS services is meant external interface by information resources. Information resource is register of every state agency or municipality, information system or database. The primary aim of each such register or IS is to ensure concrete, specific functions which the appropriate institution has to perform. IS and DB, which are used in institution are selected, developed and optimized specially for requirements of this specific institution, and they can be not oriented to possibility for others to receive information. But in the case if such possibility is intended it can be very specific and with different limitations. More detailed information about ISIS IS service is available [9].

2.2 E-service catalogue

E-service catalogue is a part of ISIS architecture. It forms the unified repository for ISIS E-services. E-service is on-line service ensured by state or municipality which is used by inhabitants, enterprises or state institutions. Applied architecture ensures possibility to provide the same e-service by use of unlimited number of entry points for example State portal, ISIS e-service portal and web pages of municipalities. More detailed information about ISIS E-service is available [9].

2.3 XML scheme catalogue

Data exchange between e-services and IS services is maintained by use of standardized and confirmed XML data structures which are included in centralized and publically accessible catalogue of XML structures. Organizational way of XML data structures is hierarchy, and data structure described in XML language can be included and re-used in other data structures as shown in figure 2. Centralized maintenance and well-considered development of hierarchy of XML data structures will ensure that each e-service or IS service will not need to have produced separate special data structure but more often used data objects (for example general personal data, general data of addresses etc.) will be repeatedly used in many of e-services and IS services. More detailed information about ISIS E-service is available [9].

2.4 Standard of metadata and e-service identification

By analogy with e-GIF, AGLS and other initiatives of e-management about metadata standard [7] set of Dublin Core (DC) elements were used [2]. Additionally XML scheme for verification of metadata elements was offered [6].
The second part of standard intends implementation of IETF RFC2141 [10] based identification system. Analysing existing options for resource identification it is formed scheme issuing unique identifiers according to URN principle what is regulated by IETF RFC4617 [1]. ISIS assigned identifier (according to URN syntaxes it is called as number) is written in the following form:

\[
\text{URN:IVIS:100001:DOC-RCM-META}
\]

Unique identifier of ISIS system has inherited all advantages of URN standard including:

- structural framework. Identifier has defined determinate structure (also in form of XML scheme) and is possible automated verification of it;
- readability. Identifier is understandable and usually includes understandable instructions for user on its source resource;
- uniqueness. Unique identifier is applied to information unit for unequivocal identification (XML schemes, documents and standards, ISIS IS services, e-services and e-service institutions);
- persistence. ISIS identifier stays unchanged from the time it is implemented by the time it is deleted. It is possible to delete ISIS code if it is useless, anybody does not use it or will not use.

Infrastructure of URN identifiers resolution has not yet established in the world scale. Global URN identifiers resolution system will be two-stage process. It will be formed as global identification register ("identifiers space") which will be able to find national, international or any other server which comprise information about registered URN and is responsible for information by definite NID (URN scheme). To such server will be re-addressed question to receive identifier resolution.

There are many functions used in URN ISIS system. With their assistance it is possible to receive information about XML scheme, IS service, e-service etc. One of ISIS URN application usage (ISIS URN resolution service) is described [9].

3 Logic programming paradigm

Logic programming is concentrated around basic mechanisms which are based on first-order predicate logic. It includes comparison with pattern, tree type data structure representation and automatic searching with rollbacks and cut offs. This limited complex of tools define powerful and flexible programming environment. That’s why logic programming is powerful tool for artificial intelligence and programming which is not based on numbers. This paradigm is based on first-order predicate logic and represents set of objects and dependencies between them. This means that for the conception of the representation of action use definition of mathematical relations or predicates. In logic programming attention is concentrated on declarative program meaning, on predicate definition not on procedure writing.

Logic programming fundamental concepts are:

- **Predicate** – truth function which is defined on concrete property or set of properties base. If object has this property (set of properties), then predicate equals to 1 (true), otherwise predicate value is 0 (false);
- **Fact** – true predicate, always true declaration;
- **Rule** – instructions, how to use to check concrete object functional precondition, to prognose functionality in concrete situation and to make concrete conclusions.

Logic programming idea is needed to:

1. Describe set of declarations using formal language;
2. Use logical development system to get a solution.

3.1 Prolog language basics

The basic concepts of Prolog language are predicate, fact, rule and goal. In the logic of predicates the relationships between declarations and objects are overviewed. The predicate structure is shown on figure 2.

![Predicate structure](image)
In Prolog language the predicate on figure 2 can be defined as: is(apple, green). IS in this case is the functor of predicate. Apple and green are the objects (or components) of the predicate. The number of components in the predicate is not limited. Predicate with point at the end is fact. The components of the fact can only consist of letters, numbers and underlines. The first letter of the component name can be only lower case letter. Fact is always true predicate in Prolog language. Fact shows that some action is took place on object. When defining with the fact relations between the objects, it is necessary to think about the sequence of the components. The Prolog rule is used when it is necessary to show how one fact depends of another facts. Usually there is the left and the right parts in the rule. The right part of the rule is called head. It is the output part of the rule. The left part of the rule is called body. It is the conditional part of the rule. Parts of the rule are separated by the symbol “:-”. Prolog language rule example is shown on figure 3.

**The rule in natural language**

Apple bark is green, If apple is green.

**The rule in Prolog language**

\[
\text{apple_bark(green) :- is(apple,green).}
\]

![Figure 3. Prolog rule example](image)

Prolog goal – the formulation of the question for which the answer should be found [16]. Usually the question is the fact which need to check. If the fact or rule (are in the knowledge base) of the goal leeds to the goal then Prolog system gives answer „yes“. That means that found fact is true, otherwise false. More information about Prolog language syntax is available [15].

The principle of the Prolog program execution – all is a tree. This principle from the very beginning is embeded in the Prolog environment. Programmers don’t need to worry about the procedural part of the program, because of the declarative principle for the code writing in the program. This means that programmer need only to describe rules and facts and define goal. After that the program itself will care about goal consummation. The facts and the rules describe in the Prolog formal language which in many cases is very close to natural language. This opportunity is available because of the first layer predicate logic. More information about Prolog language execution principle of the programs is available [13].

Prolog language can be used:

- Development of the deductive data bases;
- Sphere of artificial intelligence:
  - Development of the expert systems;
  - Machine learning;
  - And others
- Development of the relation data bases;
- Development of the games;
- Analyses of the graph.

### 3.2 Deductive data bases

Prolog language is very well known tool for solving problems related to deductive data base development. Deductive data base consist of two parts [14]:

- Part which contains facts (extencional part);
- Part which contains new rules which are based on new facts. These rules are based on extencional part and on user requests to data base (intencional part).

Deductive data base structure is shown on figure 4.
In deductive data bases recursion is very powerful tool. This tool is well implemented in Prolog. There is the direct relation between deductive data bases and knowledge data bases (are used in expert system development). Relation data bases are private case of deductive data bases. Using Prolog rules it is very easy to develop the extencional part of the data base. Using Prolog rules and goals it is very easy to develop the intencional part of the deductive data base.

4 Optimization for ISIS infrastructure

Each year there are new ideas of how to improve ISIS infrastructure. The special template for E-service development was created in 2008. The main advantage of this template that it is a compatible with ISIS infrastructure and was worked out using ISIS guidelines. A little bit later was developed ISIS URN resolution solution. Another one project is dedicated to E-service development optimization will be started as soon as possible. Last two solutions are described further in the paper.

4.1 ISIS URN Resolution

There are a lot of functions, which are executed using ISIS URN, in ISIS system. By ISIS URN is possible to get information about ISIS entity (XML schema, IS service, E-service etc.) to which this URN belongs. This information is metadata about specific ISIS entity. Metadata is based on DC elements set [2].

At present time only one authority issue such kind of URN to ISIS entities. In the future it is planned, that any authority will be able to issue ISIS URN according to agreement. This leads to the situation, where all important state entities will have globally unique identifier at state level – ISIS URN. All inhabitants will be able to get information about entities (resources), which provides specific authority. For this purpose it is planned to create ISIS URN resolution service. More information about this solution is available [8].

4.2 E-service development optimization

Different authorities (companies) using ISIS E-service development template create their own E-services. These E-services consist of IS services which are stored in IS Service catalogue. The number of such IS services growths day by day. Unfortunately some authorities create the same IS services (IS services with same input and output parameters) not using already existing IS services. The reason is that they can’t find these IS services.

The main principle of the solution is shown on the Figure 5.

![Figure 5. E-service development optimization](image)

Solution for such kind of situations is usage of deductive data bases and logic programming. This solution is the optimization for E-service development and works as follows:

1. Authority define input and output parameters for E-service.
2. Using logic programming software and deductive data base system searches for the solution.
3. If the result is positive and needed E-service can be created using existing IS services the list of these IS services is shown to the authority.
4. If the result is negative then authority can divide the E-service into subtasks and search solutions for them. This gives opportunity to develop only needed (missed) IS services and combine them with already existing.

There is a real E-service called EP00 in Latvia. It gives opportunity for inhabitants to find out the persons whose living place is declared in their property. It was the first ISIS infrastructure E-service in Latvia. It was developed, published and accessible in 2007. This E-Service works as follows:

1. Inhabitant selects EP00 on Latvian government E-service portal (www.latvija.lv) and signs in (after signing in inhabitant is authenticated and authorized for the portal; person’s identifier is person code (for example, 010290-12345)).

2. The IS service GetPersonEstateList is called. Input parameter is person code. Output parameter is the list of cadastral values. These values identify person’s, who called EP00, properties (for example, house or flat).

3. Person selects exact property for which he wants to get persons, who are declared for living in this place. The IS service GetEstateObjUniqueAddressList is called. Input parameter is cadastral value of the exact property. Output parameter information of the exact property (living place address inclusive).

4. The IS service GetDeclaredPersonList is called. Input parameter is living place address. Output parameter is list of the persons, who are declared (living) in this living place/property.

The graphical schema of the EP00 E-Service is shown on the Figure 6.

Figure 6. E-Service EP00

The described optimization (use of deductive approach for building E-services) gives opportunity quickly determine possible solutions (if they are) for building EP00. All is needed for that is:

1. Define input parameter person code;
2. Define output parameter list of persons;
3. Get list of possible solutions and manually determine the right one for EP00 creation. The solution will be the use/call of three IS services GetPersonEstateList, GetEstateObjUniqueAddressList and GetDeclaredPersonList.

5 Conclusions

The aim of E-service infrastructure creation is to implement an environment, in which it would be possible quickly and with minimal amount of programming work to implement E-services.

The created infrastructure of E-services provides:

- Increased quality and accessibility of state and municipal services (increased convenience and time economy for clients).
- Increased efficiency and examination of state and municipal institution activities (rational use of state and municipal funds).
- Ensured, advantageous and convenient service delivery procedure for clients.
- Ensured service accessibility, using various service delivery channels, which are appropriate for each client, such as ports, customer care centers, telephones, Internet, etc.

During realization of project some recommendations for usage and adaptation of SOA occurred:
• technological standard of integration has to be universal, it means that it has to be independent of different technologies of different software producers;
• integration standard has to be based on the best experience of the world and it has to comply with that what is now accepted and developed in international IT companies;
• integration software has to be scalable: with possibility to increase its performance without complicated re-programming works actually decreasing „down time“ by the minimum because this software infrastructure will be a base for implementation of e-services and in the future are planned continuous increase either in electronization of services and intensity of their usage;
• integration software has to be with producers guaranteed development in perspective;
• it is necessary to use BPEL based solutions for integrated e-service development what ensure integration of several services and performance of asynchronous processes.

In more details positive and negative aspects of ISIS infrastructure will be researched during development of Doctorate work. In author’s doctoral work it is planned to offer (work out) the new solutions which can be applied for ISIS infrastructure optimization at the State level.

References
Automated Regression Testing Using Symbolic Execution

Dominykas Barisas, Tomas Milasius, Eduardas Bareisa

Kaunas University of Technology, Software Engineering Department,
Studentų St. 50, Kaunas, Lithuania, dominykas.barisas@ktu.lt,
tomas1987@gmail.com, edas@soften.ktu.lt

Abstract. The aim of this paper is to describe a way to construct tests which validate that changes made during software evolution did not introduce regression faults. Developers usually run a new version of the program against the same set of tests. In order to achieve this goal, symbolic execution was used for test input generation and full structural code coverage. Moreover, the extension of symbolic execution was developed to increase the quality of tests. As a result, regression faults were detected in the program. The concept of the technique and an example model are presented.

Keywords: regression testing, mutation testing, symbolic execution, automated testing.

1 Introduction

During development and support phases, software is modified to enhance its functionality, detect faults, and adapt it to different platforms. Regression testing is used to identify faults that were introduced when modifying code [1].

A large number of test inputs is generated in order to cover modified parts of the code. Then the tests are executed using generated test inputs on the old and new versions of the code, differences are identified and presented to developer with the details regarding the lines of changed code and the differences [2]. The proposed approach can provide developers with detailed information regarding code coverage and various statistics.

The rest of this paper is organized as follows. Section 2 discusses related work and gives a short overview of existing solutions; section 3 introduces the concept of symbolic execution used in proposed approach. Problem description and possible issues with traditional regression testing approaches are covered in section 0. The following section defines our automated regression testing proposal. Experimental results are provided in section 6. Finally, we conclude and propose future research directions in section 7.

2 Related Work

A lot of research has been done in the area of automatic test case generation, for example an execution of various elements in the program [11] or detection of mutants [12, 19].

Test tools are used for test case execution (for example, Parasoft JTest [13]) and random test input generation. However, random test inputs may not be sufficient to detect different behavior of the new version of program.

Significant amount of research has been done in the area of regression testing in the past few years. Some of approaches [14, 15] rerun test case with the same test inputs and check the outputs of the test case against the captured outputs.

Another approach [16] generates test input set, executes them and collects the return values and object states after the execution of each method under test. The following executions retrieve the same information and check against the initially collected return values and states. Many approaches focus on testing the changed parts of two versions of a software application and takes into account changes related to method return values, object states, and program outputs.

In some cases, finding behavioral differences between two versions of program may not be sufficient and it can be expanded by predicting object state deviations of a changed program [17] or introducing mutation testing [18].

We aim to reach the following goals:

- Detect regressions faults in the program
- Reach as high code coverage as possible
- Improve test input quality by detecting mutants
3 Model Checker

As a model checker for the Java language was chosen Java Pathfinder (JPF) [4, 6] which is built on top of a custom Java Virtual Machine (JVM) and here is used for test input generation. Model checking is done via execution of Java byte-codes, an approach that allows different byte-code interpretations to be developed.

One of the model checking modes in JPF is symbolic execution [5]. Extended interpretation of byte-codes is used to work with symbolic values. Symbolic JPF checks the code for conditional branches incorporating symbolic values, then tries to find out if the branch condition is satisfied for true and false possibilities and identifies values for each branch.

There is a number of helper functions and classes available for JPF, that allow to annotate code, and develop extensions to change and monitor the execution of JPF. One of them is the ability to register Java listeners for various JPF events, for example monitor the execution of a byte-code instruction. Therefore, it allows extensions to access information used internally by JPF. The ability to annotate code and monitoring JPF’s execution is helpful for test generation.

An example of conditioned program and execution tree of the conditioned program [7] is provided in Figure 1.

```
public int sample(int x, int y, boolean z) throws Exception {
    y -= 2;
    switch (x) {
        case 0:
            if (!z)
                throw new Exception();
            else
                return 0;
        case 1:
            if (x==1)
                return y;
            else
                assert false; // unreachable
        default:
            if (x>y)
                return x;
            else
                return y;
    }
}
```

Figure 1. A snippet of conditioned program and execution tree

4 Problem Statement

In general, mutation describes the modification of a program according to some fault model. Mutation testing is the process of deriving test cases that identify as many mutants as possible. One test input covers one path of the method which may change after the modification of the code and the path will not be executed. Therefore, there will be paths that are never tested and it will cause lower code coverage. Besides, a lot of test inputs and mutants need to be randomly generated in order to cover all paths and catch the mutants. Classic mutation process and our proposed approach are illustrated in Figure 2.

- State
- State with uncaught exception
- Unreachable state
- Path
- Final path condition

Legend

- Path condition
- Final path condition
The proposed approach uses symbolic execution which helps to improve code coverage and test input quality by detecting code mutation.

5 The Testing Technique Proposal

The process can be separated into these activities:

- Path condition generation from the source code.

Figure 2. Comparison of classic mutation process and the proposed approach

Figure 3. A concept of the software testing process
• Test data generation from path conditions. Model checker extension will be developed to improve test data generation [3], which will detect mutation faults as well.
• Execution of generated test cases
• Stored result comparison with the expected results. The test case is considered to have failed in case the result does not match the expected result.

The aim is to produce unit tests because it may be run multiple times and relatively fast. Figure 3 illustrates the described approach with more details.

Proposed concept will address the following faults introduced because of:
• Modification of the application code
• Update of the packets that application is using. The functionality should remain unchanged.
• Changes of the platform

5.1 Mutation Process

Model checking and software testing isn’t the same. By proceeding from model checking to jUnit framework it was found that model checker gives an interval of variable values in order to execute concrete path of the program. However there are cases when the infinite value set is returned and only one value needs to be chosen for the path. Only one choice doesn’t always guarantee that regression faults will be detected. In order to solve this ambiguity, mutation testing will be introduced, which aims to help generating more precise test data [8, 9, 10]. This is explained in section 6.1 with more details and an example.

Main classes involved in test data retrieval and mutated test case generation are presented in Figure 4.

5.2 Limitations

There are several limitations upon this solution introduced by the symbolic execution and constraint solver. First of all, there is no such solution, which fully complies with all the principles of symbolic execution. It is still in development and currently only supports the numeric variable types. Besides, there are many restrictions such as infinite number of states resulting from the ‘for’ loop, recursion and others. However, we are concentrating more on the states returned by symbolic execution rather than addressing issues of symbolic execution.

Secondly, some of the limitations come from the restrictions of the decision engine, called library for constraint satisfaction problems (CSPs) and constraint programming (CP). Interval from -1000000 to 1000000 is valid for variable values, otherwise an error is thrown due to this limitation and is not possible to resolve (for example, restriction \( x < -2,000,000 \) trigger an error message). These issues are not addressed in our work.

Despite all the above mentioned limitations state mutation was implemented in this work, as well as state joining (program execution path generation), test data generation from the state details, expected result calculation using generated data. Because of these reasons there is no need for manual or random test input generation in order to create a mutant resistant tests. We do not have to rewrite software code and compile it for
each mutant, no need to execute both, the original and the mutated system, for each test case and check whether it catches the mutant. Moreover, we do not have to try and guess test data which protect from mutation and achieve a high code coverage.

6 Test Execution and Test Result Assessment

This section explains how the mutation is performed by giving examples and discussing them in details. Experimental results are provided and compared with different approaches.

6.1 The Need For Mutation

After test data generation we are not sure that it detects changes in the program. Suppose we have this code:

After symbolic execution two paths are found and returned:

• \((b_2_{\text{SYMINT}}[0] + a_1_{\text{SYMINT}}[1]) > c_3_{\text{SYMINT}}[0]\)
• \((b_2_{\text{SYMINT}}[0] + a_1_{\text{SYMINT}}[0]) \leq c_3_{\text{SYMINT}}[0]\)

These paths are used to generate corresponding test cases:

• \(\text{testMe}(1,0,0)\) -> Return value: \((a_1_{\text{SYMINT}} + b_2_{\text{SYMINT}})\)
• \(\text{testMe}(0,0,0)\) -> Return value: \(c_3_{\text{SYMINT}}\)

They are entirely correct test cases as all the program paths are executed at least once. However, after the modification of the program these test cases can be no longer adequate as they do not ensure that the faulty change of the program will be found. For example, suppose we had this code: "if \((a + b > c)\)"; and it was changed to "if \((a - b > c)\)" condition. Both the test with \(\text{testMe}(1,0,0)\) and \(\text{testMe}(0,0,0)\) will return a successful test execution value "Passed", although at least one of them should return "Failed" value. Both of these tests will not detect changes in the program and the possible fault.

For these reasons, we introduce mutation testing and trying to predict possible changes in the program. The main idea is that the generated test cases should fit the initial version of the program, but may not be suitable for the mutants (changed versions of the program). In other words, generated test cases will successfully pass using the initial application and fail using mutated application. In order to generate needed test cases, we do not mutate the program itself, but the expressions of execution paths. This approach has the following advantages:

• The process of mutation is simplified because we do not try to replace the original byte-code instructions with mutated instructions. There is no need to modify the software code, compile it and execute a full analysis of the model in order to get the program execution paths and new test cases.
• There is no need to compare execution paths (the initial program and the mutant), so we can combine them and get those test cases that meet the initial version of the program and would not be appropriate to mutants.

Disadvantages of the proposed technique are the following:

• We do not know what the mutant returns. The execution path is mutated, and not the program itself, therefore it may be difficult to determine what values the mutated method will return. However, this is not needed for test case generation and test execution.
• With more complex paths, especially when there are unreachable states in the initial program, it is not possible to have 100% code coverage. One suggestion for the future work could be the extension to detect unreachable code and report it.

6.2 How is the Mutation Carried Out

Once the analysis of the model of system under test (SUT) is finished and the expressions of program execution paths obtained, it can be mutated and connected to the initial expressions, as illustrated in Figure 5.
This explained how the test cases are obtained which take the mutants into account and the program changes (possible errors) are detected.

A test case construction algorithm is defined as follows:

```java
MethodsToBeTested : List of methods which should be tested
MethodsInfoList : List of collected information about methods
TestSuite : A set of returned testcases

1. MethodsInfoList ::= []
2. TestSuite ::= []
3. for each method in MethodsToBeTested
4.    MethodInfo = new MethodInfo;
5.    MethodInfo.method = method;
6.    MethodInfo.pathConditions = JPF.findPathConditions(method);
7.    MethodsInfoList.append(MethodInfo);
8. end for
9. for each MethodInfo in MethodsInfoList
10.   for each pathCondition in MethodInfo.pathConditions
11.      mutatedPathConditions = mutate(pathCondition);
12.      mutatedPathConditions = invert(mutatedPathConditions);
13.      for each mPC in mutatedPathConditions
14.         mPC = pathCondition && mPC;
15.      MethodInfo.mutatedPathConditions.append(mPC);
16. end for
17. end for
18. for each MethodInfo in MethodsInfoList
19.   for each pathCondition in MethodInfo.pathConditions
20.      TestCase = generateTestCase(pathCondition.solve());
21.      TestSuite.append(TestCase);
22. end for
23. for each pathCondition in MethodInfo.mutatedPathConditions
24.      TestCase = generateTestCase(pathCondition.solve());
25.      TestSuite.append(TestCase);
26. end for
27. end for
28. return TestSuite;
```

Figure 5. A process of test data generation
6.3 Experimental Results

Tests were executed using the following code snippet:

```java
public int testMe(int x, int y, int z, boolean k) throws Exception {
    int res = 0;
    if((15 > y) && (x + 10 < y) && (y > 10) && (y > -x + 5)) {
        switch(z) {
            case 0: res = 0; break;
            case 1: res = x; break;
            default: res = y; break;
        }
    } else {
        if (k) {
            y *= 10;
            if (x > y) {
                res = y + 3;
            }
        } else {
            throw new Exception();
        }
    }
    return res;
}
```

The application was tested three times: first with random test input generation (JTest), second using symbolic execution (JPF) which gives full code coverage and the third with symbolic execution and the extension enabled which takes mutants into account (>, <, <=, >=, ==, !=, &&, ||, ^, +, -, *, /). There are six conditions and five mutants for each of them, three ampersand and five mathematical operator replacements (6*5+3*2+5*3=51 mutants). The number of detected faults is showed in Figure 6.

![Figure 6. Test result assessment using different test inputs](image)

The number of detected faults increased from 42 to 51 in the experiment. Test results show that symbolic execution with our extension increases the number of detected possible faults using the same number of test inputs.

7 Conclusions and Future Work

This paper presented a formal technique to the regression testing process satisfying structural code coverage with a higher quality of test data.

Experimental results showed that test data generated with model checker gives a full structural code coverage which increases a number of detected faults in the program comparing to randomly generated test inputs. However, some of mutation faults still remain. This is solved using model checker extension and improved test data generation which increases test case quality and detect more mutants using the same number of test inputs.

Tasks that could be accomplished in the future:

- Combine a number of test cases derived from the different mutants into one test case.
• Create and integrate jUnit extension in test code which keeps track of how many lines of code were executed using the generated tests.
• Add extension that supports complex data structures.
• Add extension that verifies the correctness of code not only according to the returned values, but also based on the inner states of objects or functions.

References


IMPROVEMENT OF APPLICATIONS DEVELOPMENT USING SERVICE ORIENTED ARCHITECTURE

Pavel Semenchuk
Riga Technical University, Applied Computer Science Department, Meza str. 1/3, Riga, Latvia. Pavels.Semencuks@rtu.lv

Abstract. Business applications play important role in understanding and usage of newest technologies. That’s why any tendencies in their evolution attract high attention. Distributed applications are popular today. Service Oriented Architecture (SOA) is one of modern and progressive ways to develop such systems. But it is known fact: starting projects of Service Oriented Architecture implementation a lot of organizations discover that this is much more difficult task than it seemed in the beginning. There are many causes which make SOA implementation more difficult process as it could be. There is no worked out common methodology for projects’ transition to SOA. It is important to implement Service Oriented Architecture gradually. This paper gives advices of transition to SOA and promotes the idea to correct some popular errors in system’s development. In future these ways can be used as a part of unified SOA methodology. Those advices are worked out based on author’s experience in SOA implementation.

Keywords: BPEL, methodology, SOA, SOAP, WSDL.

1 Introduction
At the modern stage of the market evolution a lot of organizations already understand what SOA is and why it is necessary. But there is no common opinion for such SOA implementation, which would give benefit for the business and ensure successful functioning of SOA for many years forward. Software developers have their own (private) methodologies [1, 2, 28], which were specially worked out for the projects with SOA. These methodologies are very common and not standardized. Such situation needs deeper research of already existing methodologies for SOA and creation of unified SOA methodology.

Some companies discover that there are a certain number of organizational, technological and architectural problems which leads to the stagnation of projects in which would be possible to apply SOA. In many other companies projects of SOA implementation in real information systems do not go out of the pilot stage.

This paper is organized as follows. Section 2 describes the basics of SOA. Section 3 describes advantages of SOA, which were defined analyzing SOA architecture. Section 4 describes advices for SOA implementation in real projects. Section 5 describes the main obstacles for SOA implementation. The advices and obstacles are author’s private opinion. The common criteria for SOA based system development are described in section 6. Section 7 describes software development models. Section 8 describes current situation with SOA methodologies. Section 9 promotes the idea of new SOA methodology. Conclusions state further directions of research.

2 SOA basics
Service Oriented Architecture – architecture of information systems which is developed and based on week-binding of system parts principle. The smallest parts of such kind of systems are services. SOA is fully independent from programming languages, protocol specifications and platforms. The main reason of SOA appearance on the market is the replacement of coding of programs (from the beginning till the end) to assembly of applications from the standard parts.

SOA includes the following standards which gives opportunity to offer service oriented solutions:
- **WSDL (Web Service Description Language)** – formal standard for the description of Web services which is based on XML (eXtended Markup Language) language. WSDL service definitions provide documentation for distributed systems and serve as a recipe for automating details involved in applications communication [27].
- **UDDI (Universal Description Discovery and Integration)** – an XML-based registry for businesses worldwide to list themselves on the Internet. Its ultimate goal is to streamline online transactions by enabling companies to find one another on the Web and make their systems interoperable for e-commerce [26].
- **SOAP (Simple Object Access Protocol)** – protocol of exchange of structural information in the distributed systems. The exchange is based on XML technology. SOAP is a way for program running in one kind of operating system (such as Windows 2000) to communicate with a program...
in the same or another kind of an operating system (such as Linux) by using the World Wide Web's Hypertext Transfer Protocol (HTTP) and its Extensible Markup Language (XML) as the mechanisms for information exchange [20].

- **BPEL (Business Process Execution Language)** – language of description of business processes. The description is based on XML. BPEL for Web services is an XML-based language designed to enable task-sharing for a distributed computing or grid computing environment – even across multiple organizations – using a combination of Web services [5].

Many SOA definitions appeared during the last couple years, but all of them describe two basic moments [13]:

- **SOA** – week-binding architecture of the systems.
- **SOA** – set of the standards which give opportunity to realize SOA principles.

From the business point of view the Web service can be expressed like a realization of business process or part of this process. SOA is the relating element of all business processes for the execution of main function.

### 3 Advantages of SOA integration

When using SOA, organization gets a wide area of opportunities. These opportunities are defined by the characteristics of Service Oriented Architecture. The main opportunity is the business respond speed on the changes of the environment and the evolution of business processes with minimal expenses (for example, in comparison with CORBA).

The analysis of SOA advantages and characteristics is based on author’s experience [17, 18] and analyzing literature [13]. The main characteristics of Service Oriented Architecture are:

- **Week-binding.** Week-binding provides simple and quick adaptation of system to the changes in the structure and realization of the services. It gives opportunity on-the-fly reorganize business processes accordingly to the changes of the market (due to the week bindings between the services). Week-binding increases competitiveness because of the full confrontation of service and business process. It decreases price of implementation due to the high level of reusing of the services.
- **Modularity.** Module principle of building gives opportunity to organize graft on development, implementation and maintenance. This principle also gives opportunity for the gradual replacement of applications and hardware.
- **“SOA is not aggressive”**. SOA gives opportunity to use all investments, which were earlier invested in the information technologies, allows avoiding the reorganizing and retesting of existing systems when including them in SOA.
- **Standardized.** Platform-independence of SOA gives opportunity to choose any software and hardware. Standardization allows avoiding software lock-in, decreases complicity and fragmentation of the result from the using of closed products.
- **Technological.** One and the same technology can be applied for the wide spectrum of business problems.

Oriented Architecture – architecture of information systems which is developed and based on week-binding of system parts principle. The smallest parts of such kind of systems are services. SOA is fully independent from programming languages, protocol specifications and platforms. The main reason of SOA appearance on the market is the replacement of coding of programs (from the beginning till the end) to assembly of applications from the standard parts.

### 4 Transition to SOA and possible results

Analyzing the ideology, which is included in SOA, author can pick out two stages which are necessary for the successful transition to SOA [17, 18]:

- Creation of Web services.
- Building SOA infrastructure.

Creation of Web services can be achieved:

- Web service can be written from the beginning.
- Transformation of existing applications to the service oriented applications.
- Supplementation of existing applications with shell, which gives opportunity to work with this application like with a Web service.
- Web service can be found between already created Web services (e.g. UDDI).
- Web service can be ordered (Web services providers).

Infrastructure of SOA is an instrument of management and security both for the system at all and for each separate service which is attached to the infrastructure. Interaction of systems and applications without an infrastructure is shown on Figure 1; with infrastructure – on Figure 2.

![Figure 1. Interaction of the systems and the applications without the infrastructure](image1)

![Figure 2. Interaction of the systems and the applications with the infrastructure](image2)

In the Figures 1 and 2 cycles means the integration points of applications. Through these points interaction between systems is supported.

In the Figure 1 the interaction of the systems without SOA infrastructure is shown. Such approach needs the usage of specific relations for each interaction. To relate N systems each with each other N*(N-1) specific one-sided relations are needed. Any change in such systems is very difficult to make, in many cases it is not possible at all.

In the Figure 2 the interaction of systems with SOA infrastructure is shown. All interaction points are in the one interaction scope. Because these points don’t need any relations between the systems; they work independently. Using such approach in the worst case we need to organize not more relations than in infrastructure without SOA. Even in this case these relations qualitatively differ from relations without SOA. Relations with SOA describe interaction with abstract system. This gives an opportunity to reorganize interaction of systems without any problems. Infrastructure allows creating systems which can be complicated, scalable and ready for continuous changes. It also represents the common point of management and maintenance of all system’s parts.

5 SOA problems

Nothing is ideal. SOA is not ideal too. There are some big obstacles, which trouble successful evolution of projects with SOA [14]:

- **Full functional compatibility is not achieved.** SOA is based on standards of functional compatibility. Functional compatibility of different software products and platforms forms meaning of Web services. There are a lot of organizations which work on standards for interaction of Web services. But still, even with basic standards such as SOAP and WSDL, there are problems. In different cases these basic standards can be used a little bit different ways. To solve such kind of problems organizations should force their providers to make products which are compatible at least with basic standards of Web services.

- **There are not enough SOA specialists.** There are many kinds of architectures for information systems development. Most of the system architects usually work with one or two architectures. To be a real specialist in SOA this is not enough. To work with SOA, architect should understand corporative architecture and have good knowledge of technical, informational, business processes and data architectures. Such qualified architects are rarity.
Standards are not full, limited or badly worked out. The real problem is that standards are limited or badly developed. For example, WSDL is limited standard; WS-BPEL is badly developed standard. It is not able to provide requirements to service or contract limitations for user of service. WS-BPEL includes WSDL and UDDI standards, which are badly combined one with another.

Software products are limited. SOA creation can’t be achieved only with one software product. During SOA implementation organizations need to solve different tasks and problems with different software products. As the result organizations need to use different software products from different providers. But these providers still try to achieve functional compatibility which was described above.

6 Criteria for SOA development

For SOA development is necessary to define criteria. In author’s opinion some already existing methodologies (RUP, XP) can be taken as a criterion for SOA methodology. The criteria for SOA development are presented below [28]:

Delivery strategy. There exist three common strategies in delivering a SOA, depending on the amount of front-end analysis of the business domain and the treatment of existing legacy systems [25]. The top-down strategy is closely tied to an organizations’ existing business logic from which required services are derived. The bottom-up strategy is the opposite in that it focuses on legacy systems, and Web services are built on an as-needed basis. The meet-in-the middle (agile) strategy finds a balance between incorporating service-oriented design principles into business analysis environments without having to wait before integrating Web services technologies into technical environments [25].

Lifecycle coverage. Some proposed approaches aim to support the full SOA lifecycle, including planning, analysis and design, construction, testing, deployment, and governance activities while others limit their scope to a subset of these phases, such as analysis and design.

Degree of prescription. SOA methodologies range from the most prescriptive ones that specify phases, disciplines, tasks, and deliverables for each of them, while others provide less detail, by purpose or not, leaving room for more flexibility and tailoring of the approach depending on the project’s context.

Availability. A number of methodologies proposed by industry players such as IBM, Sun, Microsoft, and others, are proprietary and the detailed specifications are not openly available. In contrast to open methodologies for which documentation is available to the interested public, for the proprietary methodologies it is difficult to fully analyze their capabilities and to make comparisons.

Process agility. A number of methodologies suggest an agile approach to Service Oriented development in order to address risks and add flexibility to change. Yet, some others follow a more rigid approach in the process lifecycle, or do not address the issue of agility at all.

Adoption of existing processes/techniques/notation. A large number of SOA methodologies propose reusing proven existing processes like XP and RUP, A Survey of Service Oriented Development Methodologies 3 and techniques like OOAD, CBD, and BPM, seeing service-oriented development as an evolutionary rather than revolutionary step in software engineering. Also standardized notations, such as UML and BPMN, are being adopted to visually model various artifacts.

Industrial application. It is important that a methodology be validated in proof-of-concept case studies to show that it has practical applicability and to refine it based on feedback from case studies. Unfortunately, most of the existing SOA methodologies are at an early stage and have not been applied in industrial projects yet.

Supported role(s). A service-oriented methodology may support the provider view, the consumer view, or both the provider and consumer views in an integrated framework. In the consumer’s view, development is declarative and business process oriented through service composition, while in the provider’s view it is programmatic and component oriented.

7 Existing SOA methodologies

Methodology – the science that studies the methods of problem solving [9]. This is the set of methods, principles, facilities for the achievement of the goal. The methodology can be divided into two parts: theoretical and practical.

Modern SOA methodology includes two main processes [21]:
• **Service-Oriented Analysis Process.** A separate analysis is dedicated to each business process definition associated with a given service inventory. For a full definition of a service inventory blueprint, a complete top-down delivery process is carried out, comprised of numerous iterations through service-oriented analysis process steps.

• **Service-Oriented Design Process.** All of the effort put into the analysis and service modeling processes results in a collection of service candidates that establishes the starting point for service design. Every candidate definition can be used as input for a service-oriented design process. A different process exists for each of the four primary service models, but all are shaped and structured around the application of service-orientation design principles.

There is one central aspect in a SOA that drives agile development: The service model – the model of services, their dependencies, choreography, and flows [19]. There are three common service classifications that represent the primary service models [21]:

• **Entity service.** The entity service model represents a business-centric service that bases its functional boundary and context on one or more related business entities. It is considered a highly reusable service because it is agnostic to most parent business processes.

• **Task service.** A business service with a functional boundary directly associated with a specific parent business task or process.

• **Utility service.** It is dedicated to providing reusable, cross-cutting utility functionality, such as event logging, notification, and exception handling.

Some of the existing SOA methodologies are shown below:

- IBM Service-Oriented Analysis and Design (SOAD) [12];
- IBM Service Oriented Modeling and Architecture (SOMA) [3];
- SOA Repeatable Quality (RQ) [23];
- CBDI-SAE Process [15];
- Service Oriented Architecture Framework (SOAF) [4];
- Service Oriented Unified Process (SOUP) [8];
- Methodology by [11];
- Thomas Erl’s [25];
- BPMN to BPEL [6];
- Steve Jones’ Service Architectures [24].

All these methodologies doesn’t support all criteria which were defined (section 6) for successful SOA implementation. Such situation needs more common methodology for SOA implementation in IT systems.

### 8 Software development models

Software life cycle models describe phases of the software cycle and the order in which those phases are executed. The general software development model consists of 4 phases:

• **Requirements.** Business requirements are gathered in this phase. This phase is the main focus of the project managers and stake holders. Meetings with managers, stake holders and users are held in order to determine the requirements. This produces a nice big list of functionality that the system should provide, which describes functions the system should perform, business logic that processes data, what data is stored and used by the system, and how the user interface should work. The overall result is the system as a whole and how it performs, not how it is actually going to do it.

• **Design.** The software system design is produced from the results of the requirements phase. This is where the details on how the system will work are produced. Architecture, including hardware and software, communication, software design (UML is produced here) are all part of the deliverables of a design phase.

• **Implementation.** Code is produced from the deliverables of the design phase during implementation, and this is the longest phase of the software development life cycle. For a developer, this is the main focus of the life cycle because this is where the code is produced. Implementation my overlap with both the design and testing phases. Many tools exists (CASE tools) to actually automate the production of code using information gathered and produced during the design phase.
• **Testing** During testing, the implementation is tested against the requirements to make sure that the product is actually solving the needs addressed and gathered during the requirements phase. Unit tests and system/acceptance tests are done during this phase. Unit tests act on a specific component of the system, while system tests act on the system as a whole.

The general software development model is shown on Figure 3.

![Figure 3. Software development general model](image)

The traditional and widely used software development models are:
• Waterfall life cycle model;
• V-shaped life cycle model;
• Incremental life cycle model;
• Spiral life cycle model.

### 9 SOA methodology

At this moment technological aspects are worked over much better than methodological questions of projects realization with integration of SOA. Software developers have their own (private) methodologies (mainly based on RUP [16] and XP [7]), which were specially worked out for the projects with SOA. But still there is no unified SOA methodology. This disturbs the realization of unified integration system and stops many architects from transition to SOA. Methodology described above is very common. It is based on agile methods, but there isn’t any advises about exact methods for exact tasks. It is very difficult to decide which method to use in exact situation.

There are already some ideas for SOA life cycle (on Figure 4) and Web service life cycle (on Figure 5) [29]. Author of this paper consider that these parts can be taken into account for the unified SOA methodology creation.

![Figure 4. SOA life cycle](image)

![Figure 5. Web service life cycle](image)

As shown in Figure 3 SOA integration is iterative process. It means that the spiral model of development process can be applied as a base for SOA model of development process. SOA model of
development process can be applied as a base for SOA methodology (Figure 6). All these need a deep research. This research is planned as the future work on doctoral work.

The idea of SOA methodology is based on SOA model of development process and other new methods, which are not defined yet and will be included after a deeper research of already existing standards of SOA. SOA model of development process will be based on best practices of already existing models of development process (etc. spiral, waterfall and others) and combined with new original ideas, which are not worked out yet and is a part of the future research. One of the new methods, which is planned to work out for SOA methodology, will be SOA test method (the starting point of this idea can be found in corresponding works [22]). Testing is an important part of any kind of projects. Errors in the big projects cost too much (SOA projects included).

Figure 6. Idea of the SOA methodology

There are already some positive examples of new model creation which are based on previously developed models. For example, Microsoft Solutions Framework model [10] which includes best practices of spiral and waterfall models. In author’s opinion SOA needs unified methodology to make easier transition to SOA.

One of the main criteria for SOA development is life cycle coverage (Section 6). There is a lot of existing software development models (some of them are mentioned in section 8) which can be applied implementing SOA. Another one of the main criteria for SOA development is the degree of prescription (Section 6). That’s why such models as waterfall, incremental or spiral model could be applied for SOA implementation; they are enough detailed and have enough phases. As the SOA implementation process should be iterative and flexible for changes the waterfall and incremental software developments models can’t be fully applied for SOA projects. Waterfall model is not iterative and flexible. In the incremental model each phase of an iteration is rigid and do not overlap each other. In such direction spiral software development model is a good solution as it fully iterative and flexible.

10 Conclusions

There are a lot of obstacles (some of them are discussed above in this paper) for the successful SOA implementation. All of them are overcome if to act in series. The positive result of SOA implementation can give the following advantages:

- Quicker and more flexible changes of business processes.
- Decrease of expenses for the IT operations.
- Quick implementation of updates and additional opportunities of software products.

The integration is a successful relation between data, applications, processes, people and organizations. From this point of view integration is a process, not technology. That is why while implementing SOA, it is important to think about the investment which the architecture will invest in improvement of business process from which depend existence and evolution of the organization.

From the methodological point of view SOA needs an iterative approach. This is because the requirements are constantly changing. At this moment technological aspects are worked over much better than methodological questions of project realization with integration of SOA. Software developers have their own (private) methodologies, which were specially worked out for the projects with SOA. In author’s opinion, there are not enough good specialists in SOA and there is no unified methodology for SOA implementation. These disturb the realization of unified integration system. In author’s doctoral work it is planned to offer (work out) the new methodology for the development of distributed information systems which will be based on SOA principles. One of the main aspects of this methodology will be SOA testing.
References


[26] UDDI. Access via the Internet: http://searchsoa.techtarget.com/sDefinition/0,sid26_gci508228,00.html

[27] Web Services Description Language (WSDL) 1.1. Access via the Internet: http://www.w3.org/TR/wsdl


A CONTRACT-BASED APPROACH TO IMPROVING AUTOMATIC TEST GENERATOR

Eduardas Bareisa¹, Justinas Prelgauskas², Tomas Neverdauskas²

¹Kaunas University of Technology, Software Engineering Department, Studentu str. 50-406, Kaunas, Lithuania, eduardas.bareisa@ktu.lt
²Kaunas University of Technology, Software Engineering Department, Studentu str. 50-101a, Kaunas, Lithuania, justinas.prelgauskas@ktu.lt, tomas.neverdauskas@stud.ktu.lt

Abstract. Design by contract (DbC) is a well-known technique for creating reliable and safety-critical software. Contracts should be introduced into software as soon as possible. However, we believe that contracts could be exploited more, especially for improving quality of unit test suites. In this document, we present our white-box test generation tool “SYMEX” that is based on symbolic execution and is able to take advantage of contracts in code. After extending symbolic execution tool with contract interpretation feature we evaluate its impact on generated test suites.

Keywords: design by contract, unit testing, white-box test generator, executable code contracts.

1 Introduction

Contracts have been used in software development for many years. The principles were developed by Bertrand Meyer as the concept of ‘design-by-contract’ [1] where correctness requirements are expressed as a contract between a method and its callers. Furthermore, static symbolic execution was well known since 1976 and was developed by James C. King [2]. However, there were few attempts to integrate those prior mentioned ideas to improve automatic white-box test generators based on static symbolic execution with code contracts[3]. In this publication we introduce a static symbolic execution tool for .NET platform, called “SYMEX”[4]. Tool is now able to take advantage of code contracts such as pre- and post-conditions to generate better test suites. Test suites generated using our method have same code coverage ratio but have less test cases.

Document is organized as follows. In chapter 2 we introduce the principle of ‘design-by-contract’, ‘symbolic execution’ and ‘constraint solving’. In chapter 3 we go deeper into implementation details of SYMEX test generator: we present architecture and main algorithm. Chapter 4 is dedicated to experiments and evaluation of our tool. We conclude in chapter 5, providing method pros, cons and further development guidelines.

2 Background

Our test generator, based on symbolic execution can be useful in practice only if we have: a) a program to generate test inputs for; b) a test oracle that decides whether a program execution was successful for some given test inputs (method post-conditions may serve as such oracle). This white box test generation method is based on three cornerstones:

• **Symbolic execution** – to tackle state space explosion problem;
• **Code contracts** – code contracts serve as imprecise test oracle, where post-conditions are being checked as test pass/fail criterion[5];
• **Constraint solving** – to generate concrete test inputs as counterexamples to path constraints[6, 7].

We will describe those cornerstones in more detail further in this document (see 2.1. 0 and 2.3).

2.1 Symbolic execution

Symbolic execution is a natural extension of normal execution, providing the normal computations as special case. Computational definitions for the basic operators of the language are extended to accept symbolic inputs and produce symbolic formulas as output [2]. The state of a symbolically executed program includes the symbolic values of program variables, a path condition (PC) and a program counter representing next statement to be executed. The path condition is a (quantifier-free) BOOLEAN formula over the symbolic inputs. It accumulates constraints which the inputs must satisfy in order for an execution to follow the particular associated path [8]. A symbolic execution tree represents the execution paths followed during the symbolic execution of a program. The nodes represent program states and the arcs represent transitions between states.

Symbolic execution tree has the following interesting properties:

1) Each terminal leaf in the tree may have path constraint value of TRUE, FALSE or PC\text{pred}. If terminal leaf path constraint PC = TRUE, this means that method under analysis does not have any symbolic parameters that could influence execution path. If terminal leaf path constraint
$PC \equiv \text{FALSE}$, it means that the corresponding execution path is infeasible (i.e. dead code detected). If $PC \equiv PC_{\text{smt}}(a_1, \ldots, a_n)$, where $PC_{\text{smt}}$ is a satisfiable SMT formula and $a_1 \sim a_n$—symbolic arguments, there exists a non-symbolic input to the program which will trace the same program path as compared to the path, covered in symbolic execution tree.

2) Path constraints associated with any two terminal leaves are distinct. The two paths from the common root of the execution tree leading to any two terminal leaves have a unique forking node where the two paths diverge.

If we define symbolic execution for a simple integer language as described in [2], it would satisfy an interesting commutative property: the operation of instantiating the symbols $a_i$ with specific integers, say $a_i'$, and the operation of executing the program are interchangeable.

However, with today’s real world applications, it does not suffice to simplify languages to linear integer as in [2]. Programming languages are able to express non-linearity, non-determinism, have real, floating-point data types and operations. Languages have other properties which cannot be handled only by means of static symbolic execution. This is where dynamic symbolic execution (or concrete symbolic execution in some literature [9]) came into contribution.

To illustrate differences between concrete and symbolic execution, consider the simple example in Figure 1. Assume that the values of the input parameters are $x=2$ and $y=1$. Concrete execution will follow only one program path, corresponding to the $\text{TRUE}$ branch of the IF statement at line 1.

As an opposite to concrete execution, symbolic execution starts with symbolic values, say $x = \text{Sym}x$, $y = \text{Sym}y$, and the initial value of path condition (PC) is $\text{TRUE}$. The symbolic execution tree is illustrated in Figure 2. At each tree leaf, PC is updated with constraints on the inputs in order to choose between alternative paths. After executing first line in the code, both alternatives of the IF statement are possible, and PC is updated accordingly. If the path condition becomes $\text{FALSE}$, it means that the corresponding path is infeasible and symbolic execution does not continue for that path.

2.2 Constraint solving / theorem proving

After transforming branch conditions into solvable constraints, we must pass them to some SMT (Satisfiability Modulo Theories) solver. Our provided example (see Figure 1) is a very simple linear constraint.
that can be solved by simple mathematical linear programming methods and tools that implement them [10], but modern SMT solvers are sophisticated tools able to solve more than linear integer expressions. For example, string library routines can be modeled at the level of the theory of strings [11].

SMT-LIB initiative [12] has a goal to establish a library of benchmarks for Satisfiability Modulo Theories, that is, satisfiability of formulas with respect to background theories for which specialized decision procedures exist - such as, for instance, the theory of lists, arrays, linear arithmetic, and so on. There is a number of tools being developed: Alt-Ergo[13], Barcelogic[14], Beaver[15], Boolector[16], CVC3[17], DPT, MathSAT[18], OpenSMT[19], Yices[20], Z3[21], etc.

Most of the common SMT approaches support decidable theories. However, many real-world systems can only be modeled by means of non-linear arithmetic over the real numbers involving transcendental functions, e.g. an aircraft and its behavior.

2.3 Code contracts

Code Contracts is .NET Framework 4 feature that is provided via System.Diagnostics.Contracts namespace. This namespace contains classes needed to express coding assumptions in programs. The contracts take the form of preconditions, post-conditions, and object invariants. That is basically what OCL offers, but contracts are embedded in programming language (see example in Figure 3). This approach has number of benefits[22]:

- The language of conditions is just the language of expressions in the programming language used. The existing editor and IDE can be used to author the contracts. Furthermore, the IDE actually supports writing proper contract expressions by providing such features as highlighting, auto-completion, and early feedback on erroneous expressions (due to the fact that the existing language will background check the expressions as normal code).
- Refactoring tools work properly on contracts as well, e.g., renaming a parameter will rename any parameter use inside specifications as well. Contrast this to having specifications in code attributes or special code comments.
- Contracts act as checked documentation of external and internal APIs in .NET platform. They are used to improve testing via run-time checking, enable static contract verification, and documentation generation.

Thus, programmers don't have to learn a new language, a new compiler, or a new IDE, and the authoring of contracts feels like writing code.

We use contracts to improve testability. The benefits of that are as follows:

- Each contract acts as an oracle, giving a test run a pass/fail indication.
- Automatic testing tools, such as SYMEX, can take advantage of contracts to generate more meaningful unit tests by filtering out meaningless test arguments that don't satisfy the pre-conditions.

```csharp
string Compute(string str, int index, ICollection c, out int len)
{
    Contract.Requires(str == null || 0 <= index && index < str.Length);
    Contract.Requires(!string.IsNullOrEmpty(Contract.Result<string>()) &&
        c.Count > Contract.OldValue(c.Count));
    Contract.Ensure(Contract.ValueAtReturn(out len) >= 0);
    Contract.Requires(str == null ||
        Contract.ForAll(0, Contract.ValueAtReturn(out len), i =>
            Contract.Result<string>()[i] == str[i]));
    // Other code...
}
```

Figure 3. Example of code with embedded code contracts

The use of a library has the advantage that all .NET languages can immediately take advantage of contracts. There is no need to write a special parser or compiler. Furthermore, the respective language compilers naturally check the contracts for well-formedness (type checking and name resolution) and produce a compiled form of the contracts as CIL code. Previous approaches based on .NET attributes fall far short as they neither provide an expressive enough media, nor can they take advantage of compile-time checks.

Contracts are expressed using static method calls at method entry. Tools take care to interpret these declarative contracts in the right places.
# SYMEX tool

## 3.1 Architecture

SYMEX is an automated white-box unit test generation tool. The data flow in the usage context of the tool is shown in Figure 4. SYMEX reads CIL instructions from compiled DLL files, ensuring independence from concrete programming language in the .Net platform. Then, it searches for inputs in class methods, executes the code symbolically and forms a symbolic execution tree with all possible execution paths. The SMT equation parser parses data into standard SMT language and calls the SMT solver to calculate concrete values. SYMEX architecture allows using any of the SMT solvers from the SMT-Lib initiative (in this study, we use Microsoft Z3 SMT solver). In the final step, solved values and method metadata are passed to the unit test file generator and the generator outputs a generated unit test file (s). Our tool is useful in practice if we have a test oracle (or method post-conditions) and the method is at least defined for inputs (parameters).

![Figure 4. SYMEX architecture](image)

## 3.2 Algorithm

When execution is started, SYMEX looks up a PE (Program Executable) file for all classes and all methods inside. For each method, a following routine is called:

```plaintext
function ExecuteMethod(Method M)
    DO
        WHILE there are any operations left
            DO
                OPERATION = GetNextOperation(M) // Retrieves next operation to execute
                IF (OPERATION is non-branching operation)
                    ExecuteNonBranchingOperation(OPERATION)
                ELSE
                    BC = EvaluateBranchCondition(OPERATION)
                    IF BC = TRUE
                        // continue execution along "TRUE" path
                        .......
                    ELSE IF BC = FALSE
                        // continue execution along "FALSE" path
                    ELSE // BC is symbolic expression that may evaluate either to TRUE or FALSE
                        SplitControlFlow(OPERATION) // continue execution on both paths
                END
            END
        END
    END
```

![Figure 5. SYMEX execution algorithm](image)
Symbolic execution is like normal execution, but with ability to introduce additional — symbolic values into typical expressions. Let’s say we have a binary CIL operation “ADD”. This binary operation can be resolved in two ways (this depends on types of operation arguments). If one or more argument is a symbolic value, result might be either symbolic or numeric value, e.g. $ADD \ 3 \ a' = \ (3 + a')$, $ADD \ (a' - b') \ (b' - a') = 0$. However, if all operation arguments are non-symbolic values, then symbolic execution is equivalent to normal execution. Our algorithm separates two groups of CIL instructions: branching and non-branching.

### Table 1. Branching CIL operations

<table>
<thead>
<tr>
<th>Op-code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Beq</td>
<td>Branch on equal.</td>
</tr>
<tr>
<td>Bge</td>
<td>Branch on greater than or equal to.</td>
</tr>
<tr>
<td>Bgt</td>
<td>Branch on greater than.</td>
</tr>
<tr>
<td>Ble</td>
<td>Branch on less than or equal to.</td>
</tr>
<tr>
<td>Blt</td>
<td>Branch on less than.</td>
</tr>
<tr>
<td>Bne</td>
<td>Branch on not equal.</td>
</tr>
<tr>
<td>Brfalse</td>
<td>Branch on false, null, or zero.</td>
</tr>
<tr>
<td>Brtrue</td>
<td>Branch on non-false.</td>
</tr>
</tbody>
</table>

When one of branching instructions is being executed, branch condition may be evaluated to one of the following: TRUE, FALSE, some symbolic value $S'$. If branch expression evaluates to TRUE (FALSE), this means that execution will follow only the “TRUE” (“FALSE”) path. Furthermore, no control flow splitting will be done. However, if branch condition is not equivalent to TRUE or FALSE but to some symbolic value $S'$, control flow is being split such that execution is continued along both paths: $S' = \text{TRUE}$ and $S' = \text{FALSE}$.

### 3.3 Data structures

During symbolic execution SYMEX maintains a binary tree-like structure. We call it *TraceTree*.

#### 3.3.1 Trace tree

Symbolic execution trace tree defines branches in program execution together with corresponding call stack entry as well as Path Constraint. If program contains no branching operators, than this data structure transforms into a simple linked list. Every leaf node of this tree will show the end of method execution in some certain path. This leaf node of Trace Tree will have a path condition and return value associated with it. Having such data is enough to analyze every possible program execution path statically.

![Figure 6. Example of SYMEX TraceTree data structure](image)

Example trace tree in Figure 6 shows execution with 2 leaf nodes (crossed circle), 1 branch (square) that splits control flow and one branch that does not.

#### 3.3.2 Call stack entry

Every trace tree node has a stack of *CallStackEntry* objects associated with it. This object has four properties associated with it:

1) Method metadata, which enables to trace which method is being executed;
2) A stack of symbolic objects for instruction evaluation, that lets SYMEX perform symbolic operation evaluation;
3) A dictionary of method parameter metadata and their associated symbolic values, that is used by SYMEX to store and retrieve method parameters;
4) A dictionary of method locals metadata and their associated symbolic values, that is used by SYMEX to store and retrieve method locals.

Call stack entry is required to support method calls.

4 Experimental evaluation

The purpose of this experiment is to show how much influence Code Contracts have to unit test generation time, coverage and quality.

4.1 Test subjects

For this experiment we have used an open source project “Extensia” (http://extensia.codeplex.com/), which is a large collection of extension methods for LINQ library in .NET 4 platform. It can be used to complete various tasks of daily programming. “Extensia” project uses embedded code contracts to ensure code quality and validity. Tested version (2010-03-05 alpha) of this library has 750 lines of code. Table 2 has short description of tested classes. This library uses modern C# programming language features and practically used source code that can be useful in various conditions and contexts – from database to random number generation.

Table 2. Description of benchmarks

<table>
<thead>
<tr>
<th>Name of Benchmark</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extensia.Base64</td>
<td>Extensions to Base64 encode and decode binary data.</td>
</tr>
<tr>
<td>Extensia.Combinators</td>
<td>Extensions for permutations from combinatorics.</td>
</tr>
<tr>
<td>Extensia.Console</td>
<td>Console argument parsing extensions.</td>
</tr>
<tr>
<td>Extensia.Conversion</td>
<td>Extensions for value conversion.</td>
</tr>
<tr>
<td>Extensia.Data</td>
<td>SQL extensions for reading data from a database.</td>
</tr>
<tr>
<td>Extensia.Error</td>
<td>Extensions for handling exceptions.</td>
</tr>
<tr>
<td>Extensia.Form</td>
<td>Windows Forms control extensions.</td>
</tr>
<tr>
<td>Extensia.Information.Randomness</td>
<td>Extensions for declarative random number generation</td>
</tr>
<tr>
<td>Extensia.Information</td>
<td>Unsigned 1-bit, 8-bit and 32-bit integer extensions</td>
</tr>
<tr>
<td>Extensia.Infoset</td>
<td>These extensions allow the querying of XML elements and attributes across namespaces in that only the local name is used for comparison</td>
</tr>
<tr>
<td>Extensia.Logic</td>
<td>Extensions for composition of logic functions.</td>
</tr>
<tr>
<td>Extensia.Memoization</td>
<td>Extensions for function memoization</td>
</tr>
<tr>
<td>Extensia.Metadata</td>
<td>Reflection extensions</td>
</tr>
<tr>
<td>Extensia.Morse</td>
<td>Extensions for Morse code</td>
</tr>
<tr>
<td>Extensia.Numeric</td>
<td>Numerical extensions for rounding off values</td>
</tr>
<tr>
<td>Extensia.Structure</td>
<td>Extensions for IEnumerables generic type.</td>
</tr>
<tr>
<td>Extensia.Text</td>
<td>Text extensions</td>
</tr>
<tr>
<td>Extensia.Time</td>
<td>Extensions to perform high-precision timing of actions</td>
</tr>
<tr>
<td>Extensia.Transient</td>
<td>IO extensions</td>
</tr>
</tbody>
</table>

4.2 Experiment conditions

For this evaluation “Extensia” is used “at is” without any code or environment changes. Evaluation had two steps: 1) generate tests and 2) run tests. Three types of measurements were collected: “Unit Tests – without Code Contracts”, “Unit Tests with Code Contracts, dynamic execution on” and “Unit Tests with Code Contracts, dynamic execution off”. First measurement was recorded when SYMEX generated test suites from code with code contracts removed, i.e. Code Contracts were turned off during test generation and execution (McCabe’s complexity of this code was 466). Second measurement was recorded when SYMEX generated test suites from code with code contracts (McCabe’s complexity of this code was 590). Test suite was run on code with Code Contracts as well. Finally, the third measurement was collected using same test suite as in second measurement, but running it against “Extensia” sources with Code Contracts removed.

4.3 Experiment results

Test results are shown in Figure 7, Figure 8 and Figure 9.

Figure 7 shows that achieved code coverage is almost equal on all procedures. Plain symbolic execution has a little bit better coverage (48.97% instead of 41.91% and 39.96%). SYMEX performs symbolic evaluation that is basically a full control-flow analysis. If code-under-test has higher McCabe’s complexity, there are more paths to analyze. From Figure 7 we see that test generation could not achieve same high coverage for more complex code, because not all the path constraints were solvable for symbolic execution engine.
5 Conclusions

In this paper we presented a tool, called SYMEX. It is a symbolic execution engine for .NET platform, that is now able to take advantage of contracts, embedded in code. Our tool performs a white box code analysis using a constraint solver to determine relevant test inputs. Preconditions allow removing of irrelevant test inputs.
and post-conditions guide test generation as an oracle and allow detecting bugs; code contracts serve both purposes.

Software with code contracts has higher McCabe’s [23] complexity than same code without contracts. However, we showed that this increase in complexity is good for automatic white-box test generation tool. We succeeded to show that generating unit tests for code with contracts turned on, we managed to minimize test suites in such way, that code coverage was not affected. Furthermore, when dynamic code contract checks are turned on, more unit tests pass, because input validation is done by code contracts. So contracts are good for the code under test too – they make code safer by not allowing invalid inputs.

Although we were able to show that our proposed unit test generation method is practical, there is much room for improvement. First negative aspect of this method is code coverage. Even though we managed to achieve coverage of code from 39.96% to 48.97% automatically, for industrial applications this is too low. Main improvement areas could be: a) implementing dynamic symbolic execution engine to trace runtime values; b) provide a good means for class constructor abstraction as this was one of main reasons of low code coverage; c) provide a way to symbolically reason about external code (this may result in combined method, that is able to symbolically execute either manager or unmanaged code).

References

A DOMAIN UNDERSTANDING THROUGH CONTEXT-BASED FEATURE MODELLING: A RESEARCH FRAMEWORK

Vytautas Stuikys, Kestutis Valincius

Kaunas University of Technology, Software Engineering Department, Studentu str. 50,
Kaunas, Lithuania, vytautas.stuikys@ktu.lt, kestutis.valincius@gmail.com

Abstract. We propose a feature-based framework to analyse and understand the research domain such as meta-program design and evolution processes. At the core of the framework are two high-level feature-based models: the base domain (i.e., the research domain) model and its context model represented explicitly using feature notation (i.e., feature diagrams). Initially the base model is clearly separated from its context model; then, after understanding of the latter, the relations of the type required between the selected context features and the base sub-model features are identified. Such a vision enables to enrich the base model with concepts (in general, with terminology, approaches, techniques, mechanisms, tools, etc.) taken from the context domain assuming that those concepts are better understood. We have chosen software changeability research as a context of the base domain because both use changeability as a primary concern. We have presented the description of the framework at two levels: the high-level (i.e., meta-model level) and level of motivating examples (of models, programs and meta-programs). The result is a motivated formulation of research tasks and a systematic roadmap for dealing the tasks.

Keywords: meta-programming, program and model transformation, meta-program design and evolution.

1 Introduction

A well-grounded human activity, such as research, learning, design of a system, etc., starts from analysis and knowledge extraction to achieve the prescribed aim. Analysis is based on modelling, which is aiming to construct a domain model from the extracting facts and knowledge. The model enables to better understand the domain per se and facilitates further analysis that leads to the domain implementation. Though there is a variety of ways to do that, feature-based modelling prevails in research and engineering now, especially in Product Line Engineering (PLE) [1]. The PLE methodology focuses on maximizing reuse in software Product Lines (PL) (i.e., families of programs that share common assets), and mainly operates with features that can be recombined in different ways to achieve different versions of program functionality. At the core of feature-based modelling is the concept feature. In general, feature is defined as a characteristic of a system (concept, model, etc.) that is visible for an external viewer or shareholder [2]. A feature-based model is a set of features and their relationships. Typically a feature-based model is represented using the graphical notation, i.e. feature diagrams (FDs) [3]. The fundamental property of feature models (or, perhaps, models in general) is their dependency on the context, meaning that the change of the context may cause the changes in the model semantics or structure. The free dictionary defines context as “the set of facts or circumstances that surround a situation or event”. The role of context is recognized as a powerful instrument in system modelling now [4].

In this paper, we apply feature-based and context-based modelling approaches to model at a high abstraction level our research topic, i.e., program and meta-program transformation processes. The aim is to introduce software (SW) changeability research as a context to our research topic, to extract the extra knowledge from the context that is relevant to use in program and meta-program transformation thus enriching the latter domain and extending its scope of understanding. To achieve the aim, we consider the following tasks: (1) Constructing of the feature-based model for the meta-programming domain; (2) Identification of SW changeability context and representing it as a context model using FDs; (3) Identification of relationships within the topic domain and its model with the context model. Our contribution is a framework that outlines how two explicit feature models (base model, i.e. meta-programming domain model and its context model, i.e. SW changeability model) should be combined together in order to extend understanding of the first.

The remaining part of the paper is organized as follows. Section 2 describes the related work and extends the motivation of our approach. Section 3 analyzes the proposed framework to motivate and deal with the formulated tasks. Section 4 presents the feature-based meta-programming domain model the R-relationship model (meaning REQUIRE-based model). Section 0 provides SW changeability research model as a feature-based context-model and relations between the context and based models. Section 6 outlines advantages and disadvantages of proposed framework. Section 7 formulates research tasks within the introduced framework. Finally, Section 8 presents conclusions.
2 Related works

The aim of analysis is to provide a general understanding of the topic by presenting the definitions taken from the literature, motivating the topic through analysis of the most essential works that are relevant to our paper. Thus we categorize the selected works into four categories: 1) feature-based modelling; 2) context-based modelling; 3) meta-programming and 4) software (SW) changeability.

1. Feature-based modelling is due to the early seminal work [2] in which the FODA (Feature-Oriented Domain Analysis) method and FDs as a part of the method were introduced. Over two decades there were several proposals to introduce others related methods (e.g., FORM [3], meaning Feature-Oriented Reuse Method, SCV- analysis [5], meaning Scope-Communality-Variability). FDs as a notation to represent the result of modelling, i.e. the model of a domain under consideration, have also evolve continuously and experienced many changes (see, e.g., [6]). Now feature-based modelling is at the core of PLE [7]. The importance of the feature concept can be conceived, for example, from the analysis of definitions of the term. Since in the software engineering literature there is no consensus on what a feature is, we deliver some definitions of the term. A feature is: (1) End-user visible characteristic of a system or a distinguishable characteristic of a concept that is relevant to some stakeholder [2],[8]; (2) A logic unit of behaviour that is specified by a set of functional and quality requirements [7]; (3) Qualitative property of a concept [9].

A FD specifies feature types (mandatory, optional, alternative) represented as nodes of the tree, and relationships among the features (more about the notation can be learned from [6]). Relationships are represented within the tree in two ways: 1) as a parent-child relationship and 2) as constraint relationship among nodes (typically among feature variants, i.e. terminal nodes on the tree) derived from different parents. Note that this is not the rule. A set of mandatory features and their relationships can be treated as commonality, while the rest types and relationships are treated as variability within a domain to be modelled [5].

2. Context-based modelling is another branch of modelling used, for example, for knowledge management [4], ubiquitous computing, etc. As there is an extremely wide stream of research on the topic, we restrict ourselves on two lines only: the importance of the topic and connection of the topic with the feature-based modelling. Dey et al. [10] give one of the widely accepted definitions of context and defines it as “any information that can be used to characterize the situation of an entity (i.e., person, place, object, or application, etc.)”. Ubayashi et al. [11] define context as “an external or real world factor such as the usage environments that affect the system behaviour”. This paper introduces context-based model that is described explicitly using FDs for solving their task to build reliable embedded software. Lee and Kang [12] identify that the usage of context is a key driver for the feature selection. Both papers actually motivate usefulness of combining feature-based modelling and context-based modelling. The works support our vision and approach used in this paper. A. Van Duersen [31] proposes a textual language called Feature Definition Language (FDL) to specify feature models.

3. Meta-programming (MPG) is a technology for the automatic program construction. Though this domain is not new and has a long history (see [13], for extensive review of the topic), the domain is still under intensive research, where several directions can be identified: generative-programming [14], aspect-oriented programming [15], generic programming [33], feature-based programming [32]. Our approach is called structural heterogeneous programming as: 1) it is based on using meta-language (ML) (dedicated or GPL in the role of ML) and 2) it extends the pre-processing concept in the mode of structural programming [13]. Meta-programming is defined as “a process of manipulating on programs” (or their parts) as data (though there are other definitions [13]). The definition should be conceived as changes (modifications, transformations) according to a well-defined scheme. It can also be conceived as a generalization of program instances through introduced transformations. In that aspect, meta-program is a collection or family of the related program variants (instances). The meta-program development process can also be defined as an ‘encoding of anticipated domain variability’, whereas the encoding is done using heterogeneous meta-programming techniques.

4. Though SW changeability is researched in the variety of cases, however, simply its context can be outlined within the software lifecycle phases (i.e., design and maintenance & evolution). According to the standard ISO/IEC 9126 [34], SW changeability is regarded as an important sub-characteristic of maintainability. The term software evolution is to be understood in the context of continuous program change and empiric Lehman’s laws of evolution [16]. On the other hand, the need for the SW changeability studies is also due to the fact what Rajlich call “paradigm change in software engineering” [22]. He claims that the old waterfall paradigm which is based on freezing requirements for the duration of software development are not further working due to requirements volatility. The new paradigm addresses this shortcoming by emphasis on software evolution, thus opening new topics into the forefront of software engineering research. As Boehm observes in [17], now the nature of software evolution is shifting to ‘a continuous process, in which there’s no neat boundary between development and evolution’. This paradigm shifting is due to the tremendous changes in technology and competition. Furthermore, analysis of changeability-related publications allows to formulate some important observations: 1) changeability research has indeed a wide context, including meta-programming and program transformation [23]; 2) there is a large amount of factors either influencing or affecting SW.
In analyzing SW changeability research, we were able to find the only three papers which provide change-based taxonomies either explicitly [18],[19] or implicitly [20]. For example, Buckley et al. taxonomy [18] focuses on the how, when, what and where aspects of software change. Changeability as a property can also be understood through the stage-based software evolution model [21]. The model describes five stages: 1) initial development that creates the first version; 2) evolution stage to improve the version due to requirements changing; 3) servicing stage to perform small patches; 4) phase-out stage, where no more servicing is provided and the users work around the known deficiencies; 5) close-down stage that discontinues the software use. In the context of stage-based model and paradigm change, Rajlich [24] introduces incremental change (IC) design as a research topic. Though many issues of the IC design are open and require additional research, the paradigm is described as a process that includes concept location, impact analysis, change propagation and refactoring.

As a result of analysis, we can conclude that terms, variability and changeability, reflect and express two different sides of the same thing. The first relates, at a higher extent, to the program or meta-program development phase, while the second actually relates to the use and evolution phase. Variability, perhaps, can be treated as an anticipated or easily predictable change, while changeability encompasses all aspects of change, anticipated, predictable and non-predictable. What is important to emphasize is that programs/meta-programs have tendency to evolve at the use phase and evolution is realized through changes. Such knowledge gained through analysis enables us to identify that: 1) the SW evolution can be seen as a context to changeability research; 2) the latter can be seen as a context to meta-programming and program transformation research. In other words, the context can be seen as hierarchic structure. Section 3 is based on these findings.

3 A framework to deal with context-based feature models

The proposed framework describes three things: 1) the way how the context can be introduced in higher-level modelling and analysis; 2) an extended motivation of the context-based feature modelling; and 3) the structure of the framework.

3.1 How the context can be introduced in the feature model?

Knowing the context we can more easily understand a topic under analysis. This important observation was known and continuously used for a long time, for example, in learning and knowledge gaining [25]. In analysis (e.g., of a domain, system, topic, etc.), the context can be introduced either implicitly or explicitly. In our view, what form of the context to use largely depends on the following factors: aim of the analysis, the type and the scope of a domain and the scope of the context of that domain? Very often, if the context is very clear or narrow, there is enough to use the simplest form, i.e. to specify the context simply stating: “we analyze the model (topic, etc.) in the context of ...” without the explicit representation of the context within the model. The explicit representation of the context is more powerful.

The explicit context can be introduced, for example, within feature models, in two different ways: 1) either as a part of the model (for example, as the higher-level features with respect to the domain features within the model) or 2) as two separate models (domain feature model and context model, which is also expressed through features). The latter version in PL-based modelling was introduced only recently due to [11]. The context line separates the two models and at the analysis stage (more generally, at the domain engineering stage), there is the only one-way direction to express the relationship between models: from the context model to the domain model. In other words, the context model is treated as a higher-level model with regard to the domain model. However, at the application implementation stage, both models are to be integrated in somewhat way. In this paper, the term domain should be understood widely, i.e. as a set of systems (the PLE view), as a system or application, as a research topic expressed through some characteristics (features), as a set of related components.

Yet one aspect should be explained. As a majority of feature-based approaches are related to PLE and application software development (this can be learnt, e.g., from conference proceedings or other references) there might be raised the following question: is the approach (feature modelling) relevant and beneficial to apply in other domains, such as modelling of some research topic (meta-programming program transformation in our case)? The answer is given in the next sub-section.

3.2 Extended view on the use of context-based feature modelling

The extended view on the use of context-based feature modelling is motivated by the following observations: (1) The slightly different aspects of the concept feature definitions (see above Section 2.1, in essence each of them extends the vision of feature modelling at least at the conceptual level); (2) Treatment of feature models as a form of the knowledge representation (though as a week form); (3) Efforts to extend the expressive power of FDs for knowledge representation by combining this notation with fuzzy logic [29] and ontology-based approaches; (4) The use of FDs in e-learning to specify learning objects at a higher abstraction level; (5) The role of context in different disciplines and case uses. For example, Lee and Kang [12] state that
“usage of context is a key driver for feature selections”, meaning the recognition of combining the context-based and feature-based modelling.

As a result, we formulate the feature definition to better understand our approach in the following way. **Feature is a knowledge unit (concept) used to understand a topic through modelling (i.e. constructing and analyzing high-level models).** A domain under analysis may have multiple contexts; these contexts may be related, thus making up a hierarchy of contexts. For example, changeability features such as artefact (in our case a model or a program), change type (e.g., addition, substitution of features, etc.) are related with design methodologies (e.g., conventional that is based on waterfall model, or change-based [26]). And this context hierarchy is influential to such a domain as meta-programming.

### 3.3 Structure of the framework

The framework is based on the model-driven view to program (meta-program) understanding [20] and outlines the following aspects: (1) Specification of a higher-level model, i.e. meta-model of a base domain, i.e. domain under consideration, which is expressed through features; (2) Identification of a context model (also expressed through features) for the domain under consideration; (3) Instantiation of both the domain meta-model and its context model aiming: (a) to derive a concrete model for the investigation and (b) to select context features from the context model, which are influential to the analysis of the base model; (4) Identification of a relation between the selected context-based features and features of the instantiated domain to be analyzed. The domain and its scope for analysis should be selected first. It depends, for example, upon the activity one is aiming to carry out (in our case to perform research). The introduction of the context information depends upon the domain and previous knowledge or knowledge gained from literature analysis.

Some definitions are important to understand the use of the framework. **A feature-based model** is the one that consists of two kinds of elements (features and relations), where features are represented by nodes (i.e., boxes with marks meaning the kind of features) within a FD and relations are represented either by branches (parent-child relationships) or by constraints of type REQUIRE and XOR, meaning constraining relations among feature children having different parents. Feature-based meta-model is the model which describes a domain as generally as possible and contains within model instances (variants) that can be obtained through instantiation. **Instantiation** of a feature model is a form of model transformation that includes: 1) selection of features from its meta-model and 2) decomposing of the selected features into sub-features until variant features are derived. **Variant feature** is the feature value beneficial for use or understanding. **Partial instantiation** is the transformation in which the only part of features from its meta-model (model) is selected and variant features are derived from the selected features. Further, we select structural meta-programming as a case study for analysis.

### 4 Structural meta-programming as a research domain and its feature-based model

In general, meta-programming is “a manipulation on program as data” [13]. Structural meta-programming is the one, which manipulates on program with structural programming in mind. The definitions do not provide information to understand the domain in detail. To analyze structural heterogeneous meta-programming (SH MPG), we introduce the feature-based model (see Figure 1). The model extends the given definition essentially. The model is defined by four high-level features that describe: language aspects, algorithmic aspects (i.e., operations for manipulation), data aspects and model-based aspects. All these features are mandatory and they are further decomposed into smaller ones. For example, the feature algorithmic aspects reflect manipulations that are expressed through three sub-features: operation (e.g., such as assignment), condition, and loop-based ones (structural programming view). The sub-feature operation is mandatory because it is operation sufficient for describing the simplest manipulation on a program (e.g., a linear structure), whereas the remaining sub-features are alternative.

As the model describes a large part of the base domain it can be seen as a meta-model of that domain. To be useful, the model is to be instantiated first, for example, through the introduction of some context information. Instantiation enables us to construct motivating examples in order to explain and understand the topic in more details. Let introduce the following variants of features, when the meta-model (Figure 1) is instantiated: (1) ML is a Dedicated ML; (2) DL is VHDL; (3) DPI (i.e., DP instance for manipulation is the two-input AND-gate model described in VHDL, see Figure 2); (4) Changes are described as adding of a new functionality (i.e., OR function) and adding any number of inputs, see Figure 3, a).

The implementation example given in Figure 3 explains the essence of the approach only, but it does not describe the process of transformations through changes, though these changes were introduced implicitly as context information. For example, the application domain (i.e., the gate domain in our motivating example) was introduced implicitly in order to understand meta-programming through analysis of its feature-based model. For a deeper understanding, reader needs to read carefully comments within specifications: @ for and @sub (meaning substitution) are meta-commands of the ML.
Note that constraints of the type require are not shown in the model (Figure 1). Some of them we describe using FDL [31]:

MI: require MetaParameters; DPIInstance require DSL; MB require AlgorithmOperations; MetaParameters require DML; etc.

The instantiated and implemented model instance is a meta-program given in Figure 3, b.

5 Context model to extend understanding of meta-programming

Context information, such as the gate application, has been already introduced implicitly in our model and motivating example. As, in fact, context is much wider for such a domain as meta-programming, it is reasonable and beneficial to specify context information as the explicit model. As it was obtained through the literature analysis, both domains (MPG research and SW changeability research), in essence, deal with changes, though from the different perspective and intention, we accept SW changeability as a context to extend understanding of meta-programming per se. A simplified SW changeability model is represented in Figure 4 as a feature model. At the core of the model is Buckley et al. taxonomy [18]. However, it was simplified and adopted to our needs. In fact, the context model is hierarchic for such a domain as MPG. Furthermore, SW changeability has its own context. For example, changeability is discussed in the context of maintenance and evolution [30] (e.g., corrective changes, perfective changes, adaptive changes, Lehman’s laws), as well in the context of design
paradigms (design-for-change [27], incremental design [28], etc.). In this paper, however, we restrict ourselves and analyze explicitly the changeability context only.

Figure 4. SW changeability as a feature-based context model with respect to base domain - Degree of formality

How context-based model extends its base model

First, the context model should be instantiated. We will do that for some context features (TimeOfChange, Artifact, DegreeOfAutomation, DegreeOfFormality, ChangeType) using FDL [31]:

- **TimeOfChange**: one-of (Design, ?Evolution)
- **Artifact**: all (Program, Model, MetaProgram)
- **DegreeOfAutomation**: one-of (NonAutomatic, ?SemiAutomatic, ?Automatic)
- **DegreeOfFormality**: one-of (Informal, ?SemiFormal, ?Formal)
- **ChangeType**: any-of (Addition, Substitution, Deletion)
- **Impact**: all (ConceptLocation, ChangePropagation)

Note that the approach can be applied not only to the meta-program design but also to its evolution. In the latter case the base model should be transformed through changes first as it is illustrated in Figure 5.

Figure 5. Feature model extension through changes and transformation

Figure 6 illustrates the result of transformation of the meta-program into another version through a partial changes of the model (only two operations NAND and NOR are added). The partial changes are located with the meta-interface only and there is no change propagation.

Figure 6. A motivating example to explain meta-program evolution through partial changes of base model
6 Evaluation of the framework

Advantages are: (1) The proposed framework gives a unified view to the base domain under analysis and connects it with the context model represented explicitly; (2) The explicit context model enables to introduce formalism and in this way: 1) to better understand the base domain per se; and 2) to create a possibility to automatically (or at least semi-automatically) implement the domain; (3) Feature models within the framework are universal and can be applied to model and understanding of any base domain because the models can be understood intuitively; (4) The framework brings a systematization for analysis of the research topic. Therefore, it can be seen as a roadmap for the research activities to be performed. (5) The framework supports the model-driven view well, thus it well-suited for model and program transformations.\[...

Disadvantages are: (1) As the context model has a hierarchical structure, it is not an easy task to identify the model’s scope that would be relevant to the base domain, thus the construction of the model requires a great deal of knowledge and efforts; (2) As context is introduced usually intuitively a different interpretation of it is possible, thus additional difficulties may arise; (3) Though the FD notation is easy to grasp and is well-suited for the human interpretation, semantics of the graphical notation is not yet well-defined;\[...

To overcome some of disadvantages, such as a removal of semantic discrepancies, we suggest to use FDL [31] as a textual feature specification language. The latter representation is better suited for the computer-based interpretation. The difficulties in constructing of an explicit context model can be diminished, for example, by decreasing the context scope, i.e. excluding and representing explicitly only the most essential (influential) features of the context, while the remaining ones introducing implicitly. For example, in our view, the analysis of meta-programming tasks, such as meta-program design and evolution processes can be well understood and dealt with through such features as concept location and change propagation that are well known in SW evolution/changeability research, thus are introduced from the context model. As it can be conceived from our motivating examples the change propagation task within the meta-program is a complicated task (comparing with the task when change object is a program). However, such task is much easier to grasp and understood at the model representation level (see, e.g., Figure 5). As a result, it makes significant considering the reverse transformation tasks, i.e. extracting the model (if it is yet unknown) from a meta-program specification. It is reasonable first to make changes in the model and then to move to changes of the meta-program through solving the forward transformation tasks (model - to - meta-program). Combining together forward and reverse transformation tasks (i.e., transforming a feature model into meta-program and vice versa) through changeability features may form a well-grounded foundation to understand and deal with the meta-program development and evolution processes systematically.

7 Research tasks

The basic tasks to be considered are as follows: (1) The development and investigation of the product-line feature-based models. The difficulties for selecting a denotational model can be diminished, for example, by decreasing the context scope, i.e. excluding and representing explicitly only the most essential (influential) features of the context, while the remaining ones introducing implicitly. For example, in our view, the analysis of meta-programming tasks, such as meta-program design and evolution processes can be well understood and dealt with through such features as concept location and change propagation that are well known in SW evolution/changeability research, thus are introduced from the context model. As it can be conceived from our motivating examples the change propagation task within the meta-program is a complicated task (comparing with the task when change object is a program). However, such task is much easier to grasp and understood at the model representation level (see, e.g., Figure 5). As a result, it makes significant considering the reverse transformation tasks, i.e. extracting the model (if it is yet unknown) from a meta-program specification. It is reasonable first to make changes in the model and then to move to changes of the meta-program through solving the forward transformation tasks (model - to - meta-program). Combining together forward and reverse transformation tasks (i.e., transforming a feature model into meta-program and vice versa) through changeability features may form a well-grounded foundation to understand and deal with the meta-program development and evolution processes systematically.

8 Conclusions

We have identified SW changeability research as a key factor (among others) to form an explicit context model for the extended analysis of the base domain (i.e., research in model and program transformations that are based on the use of structural meta-programming). We have proposed a conceptual framework for the analysis, which is based on using both the base domain model and its context model. We found feature-based concepts as a relevant abstraction to describe both models at the high abstraction level. The explicit feature-based context model enables to better understand the base domain through the introduction of new terminology (e.g., concept location, change propagation, etc.), approved approaches and techniques (e.g., incremental change, perhaps with adaptation). It is expected that the framework will systemize the research activity and the use of the introduced formalism will contribute to the increase of a degree of automation in the model and program (meta-program) transformations.

References

TECHNOLOGY FOR LIGHT-WEIGHT ABC IN SME

Anita Finke, Ligita Businska
Riga Technical University, Department of Systems Theory and Design, Meza 1/4, Riga, Latvia, anita.finke@rtu.lv, ligita.businska@rtu.lv

Abstract. During the last 20 years most of the enterprises are adopting modern technologies – to excel in market, and to reduce costs. ABC is a method that helps to manage the costs in enterprises and to manage and understand processes of the enterprises. Traditional ABC requires an information system for finance and book-keeping. In SMEs such systems are usually not available. Therefore to do not put additional burden on SMEs book-keeping and information processing, new ways of use of ABC should be considered. Making connections between processes, cost tables and cost estimates we propose a light-weight ABC usage method, which can be implemented by using multiple non-commercial tools.

Keywords: Activity-based costing, Small and Medium Enterprises, Business Process Management.

1 Introduction

ABC method is mostly used in large companies, but in small and medium enterprises (SMEs) ABC is used in special areas, for example in wine industry [1] or cook-shops [2]. Usually when SMEs start using ABC, they change their book-keeping system to make ABC perform effectively.

Activity Based Costing is an untraditional costing method created by Cooper and Kaplan (in 1988) [3]. ABC can be considered not as a method of costing, but as a technique for managing the organization better [4]. ABC can be described as a tool used for implementing ABM – Activity Based Management [5]. This method creates objects of costs by adding the overhead costs to certain activities. ABC is used in estimation of product prices as well as in the management of quality [6]. The conditions of ABC define that the estimates include all the processes related to the production of a certain product or service, starting from the purchase of raw materials up to the moment when the result reaches the client [7].

ABC is classified as an analytical costing method [8]. Overall there are 4 divisions mentioned in literature: intuitive costing, analogical costing, parametrical costing, and analytical costing. The goal of the use of ABC method is to facilitate decision making (e.g. whether to start the production of a new product) as well as to analyze the current situation – the performance of enterprise or organization, the production of goods.

In Section 2 related work is shortly described. In Section 3 light-weight ABC is proposed. In Section 4 conclusions and future work are described.

2 Related work

There are many different issues considered by ABC research papers and applications. ABC method is used in various industries, but mostly in large enterprises. Increasing competition causes more attention on how to manage and organize processes and costs in enterprises. ABC method is used even in formation and determination of judicature’s costs and other law court costs [9]. In [10] some important points on advantages of ABC are mentioned: improves accuracy and relevance of product costing; provides timely cost information suitable for decision making.

Many authors describe problems and solutions in ABC enforcement in enterprises [5][11][7], for example data gathering and model building [12]. But most of these researches describe ABC in large enterprises [7]. Several papers [12][13] reveal discussion about business process reference models use to support business process modeling for ABC. Some of authors [10] point to shortcomings of ABC, for example, doing little to change old management behavior; not driving companies to change their fundamental views about how to organize work and to satisfy customers efficiently; requiring additional effort and expenses to obtain information needed for analysis.

Analysis of related works results in some important adjudgements, for example, ABC improves accuracy and relevance of product costing [10]. It’s a big and important issue when considering feasibility of ABC for SMEs. Experience of adapting and using ABC in large enterprises can be used developing ABC for SMEs. When we see situation and scenarios of ABC use in large enterprises, we can more easily develop scenarios for future ABC mode for SMEs.

3 Light-weight ABC or SME

3.1 Activity based costing adaptation model

As it was mentioned above, ABC is used mostly in large enterprises. While implementing this method in large enterprises their book-keeping system is adjusted or changed in order to use ABC successfully. The
management processes are reorganized, and the staff is trained. This process is difficult and demands resources; see table 1 – Traditional ABC system adoption. In case of SME large expenses are not advisable. Therefore the light-weight technology was invented to achieve cheap adoption and easy use of ABC. See light-weight ABC system adoption in table 1.

Table 1. Traditional and light-weight ABC system adoption

<table>
<thead>
<tr>
<th>Traditional ABC system adoption [18]</th>
<th>Light-weight ABC system adoption</th>
</tr>
</thead>
<tbody>
<tr>
<td>Define the scope</td>
<td>Start to use light-weight ABC</td>
</tr>
<tr>
<td>Establish the goal of the project</td>
<td>Read user manual and learn how to use the system</td>
</tr>
<tr>
<td>Form a project management team</td>
<td>Prepare data of costs</td>
</tr>
<tr>
<td>Do an initial feasibility analysis</td>
<td>Create business process models</td>
</tr>
<tr>
<td>Chose one leader who possesses leadership abilities with knowledge of ABC</td>
<td>Fill activity cost tables</td>
</tr>
<tr>
<td>Form steering committee</td>
<td>Create report</td>
</tr>
<tr>
<td>Train – Management, Implementers and Users</td>
<td>Analyze the report</td>
</tr>
<tr>
<td>Convert general ledger to a data set that can be utilized by the ABC project management team</td>
<td>Take decision</td>
</tr>
<tr>
<td>Separate activities in to primary or secondary ones</td>
<td></td>
</tr>
<tr>
<td>Develop real working ABC in enterprise</td>
<td></td>
</tr>
<tr>
<td>Start to use ABC and put information into it</td>
<td></td>
</tr>
<tr>
<td>Analyze results</td>
<td></td>
</tr>
<tr>
<td>Take decision</td>
<td></td>
</tr>
</tbody>
</table>

As seen in table 1, in order to implement the traditional ABC in an enterprise the financial as well as human resources are required; the process usually lasts for 4-6 months [18]. For a SME it is too long because of the additional costs that arise during this period. In case of SME it is advisable to use minimal amount of resources and time. The second column of the table 1 shows phases that include the acquiring of the method and the L-W ABC system, implementation of it and gaining the results. These phases can be carried out during a few days or even during one day; it is easy and does not require additional costs from the enterprise.

The phase of traditional ABC – formation of the system for cost converting from the general ledger to the respective format of data – is not carried out. The amount of data in SME is comparatively small. And the fact that light-weight ABC system is used by the management of the enterprise reduces the necessity for a specified accounting system because the manager can use the method manually by following the existing financial and book-keeping data as well as by monitoring the practical progress of the process.

3.2 Light-weight ABC

The costs in ABC model are created by attaching resources to certain activities that characterize a certain process in the enterprise. By attaching these resources to the activities we get a cost object. A cost object is anything for which the cost data are desired including products, product lines, customers, jobs, and organizational subunits [14]. For the purpose of assigning costs to cost objects, costs are classified as direct cost and indirect cost [14]. The traditional cost forming model can be seen in figure 1. In light-weight ABC (L-W ABC) this structure is preserved while additionally modeling processes that are detailed in activities and to which the resource costs will be tied.

As it was mentioned above, the traditional ABC model is shown in figure 1. But the structure of the L-W ABC model is shown in figure 2.

As seen on figure 2, the business processes lead us to the costs where the linkage between the activities and resources is preserved as in the traditional ABC model. In L-W ABC model we get cost objects when we develop a business processes model and then link costs to the process. In traditional ABC model we identify all activities (mostly it’s an activity list), and create a cost table, then resource table from which the resources will be taken and added to activities. These processes of identifying all activities are built during business process modeling.

Business process modeling and management (BPM) currently is very topical. Business process management (BPM) is a management approach focused on aligning all aspects of an organization with the wants and needs of clients. It is a holistic management approach [15] that promotes business effectiveness and efficiency while striving for innovation, flexibility, and integration with technology. Business process modeling (BPM) in systems engineering and software engineering is the activity of representing processes of an enterprise, so that the current process may be analyzed and improved [16]. This article will illustrate proposed approach by business process of a small dairy farm.
The spectrum of available non-commercial tools at the moment is very wide; these tools can be especially effective in small projects. Also open-source tools [17] as ARIS Express, BizAgii Process Modeler, Process Maker etc. are available. For calculation and tables there are open-source tools as OpenOffice spreadsheet and others. The goal is to find a scenario that would describe the performance of the system, and the links between the tools in order to provide wholesome performance of the L-W ABC system. It is possible to find such a scenario, and that provides a possibility to use ABC in a different way than before.

![Figure 1. Traditional ABC model [10]](image1)

![Figure 2. Light-weight ABC model](image2)

L-W ABC solution should provide a simplified performance of the user’s actions, and valuable results for the analysis. The user is offered guidelines on usage of ABC and the system; these guidelines include models of standardized processes that can be modified optionally in accordance to the requirements of the user. It should be noted that while implementing L-W ABC it is advisable to work with a single field at a time, e.g. dairying. Formation of standardized models for several fields at a time can be a labor-consuming task. In figure 3 L-W ABC system is an L-W ABC specified functions in tool/tools.

In L-W ABC systems implementation, it is necessary to define standard business process models, where business processes suit the subjacent business process with defined costs included in the current industry. Those standard business process models with defined costs will perform instructor’s role for user to let them understand how to form the business process model and how to work with this system. It illustrates also how “standard” processes look like in this industry. L-W ABC systems and user action scenario is shown in figure 3. There is a difference in scenarios, when the user chooses to use standard business process model or chooses to create a new model. Users in this process need simple visual user interface with most important functions – delete, save, etc. That offers an effect of an easily understandable system.

ABC system in figure 3 shows a technological solution: how non-commercial tools need to perform in order to implement the idea and goals of ABC. ABC system is a compilation of technical activities for the tool. There is a scope of functions that are to be realized, if we want to have L-W ABC as the result. Figure 3 shows that user has to perform only a few steps to achieve the result, all technical and informative work is done by the L-W ABC system of the tool.

While creating BPM the system automatically forms the tables of corresponding costs. The system provides a cost table for each activity as well as the connection between the activity and the other cost tables corresponding to the links of BPM activities (see figure 4).
Figure 3. Light-weight ABC systems and user work model

Connection between BPM model and the database tables is dynamic. When the user creates a new process or activity, system creates new record in database and offers the most appropriate data for the user, in order to fill all the necessary information. It is required in order to estimate precise total costs, and to handle separate phases while preserving the overall context.
The transition to the cost tables is done graphically – the table is opened by double clicking on the name of the process/activity. Accordingly the return from the table to the process model should be provided. Therefore there is an impression that the user is working with small graphic tables.

In case of the standard model the cost tables are formed in shape that reflects the space for the standard costs and resources, and the user inserts only the cost sums and units.

Formative model of costs collection is illustrated in figure 5. Existing patterns or standard business process models include self defined costs. For defining a new cost, it will be saved in internal database and when the user fills cost table next time this cost will be available using drop-down function.

In the report user can see a united costs table and the results of analysis. User generates the report when process model and all costs tables are completed. After report is generated, user can analyze – which process consumes most of the financial resources and so on. The user can make a decision to change the business process model or cost tables. If he chooses to do it, he goes back and does it. When he generates a new report, he can compare two reports. It’s very helpful when making important decisions in process control or financial questions.
To show full view of costs, system must divide the costs in direct costs and indirect costs, to let the user make better decisions in the enterprise management strategy. But to do it, all traditional costs types in cost collections need to be indentified and defined. Unfortunately, in traditional ABC system they use only indirect costs, because it’s more difficult to divide them between processes. In this model we take both – direct and indirect costs, to give an exact product prime cost. To know their costs and to manage them SMEs don’t use many different tools. They need one and complete tool to identify costs and to manage the processes. Therefore it was decided to include direct costs into L-W ABC.

The first goal in this research is to create L-W ABC in SMEs model, which gives instructions, and show how this L-W ABC works and what functionality for SMEs it gives. This model is created and allows us to discuss the details of user instruction and system usage.

During the project, some tests of tools were taking place, for example a test of Open Office Base. This tool gives a good opportunity to model a database with calculations, but it does not support modeling options. To show the results, we used the opportunity to model data base in Open Office Base; to model a BPM, we used ARIS Express tool. The question what technology should be used to connect tools, for example Open Office Base and some modeling tool, remains open. The result of this project was a prototype, which shows how this idea works – what will the user do to find the cost of a product or service.

4 Conclusions

Technology for light-weight ABC in SMEs is a simple ABC method, which is more simple than traditional ABC in use and does not need a lot of financial resources to be adapted in SMEs. It gives the users a possibility to work with processes in graphical mode and with only one cost table at time. A traditional ABC method is very complex and requires a special knowledge from the users. Light-weight ABC method is for enterprises owners or managers, who know the processes and need to make strategic decisions.

This kind of technology is a step forward, because it helps SMEs to manage enterprises better and let them think about enterprise management. Today it’s very usual that SMEs do not use economical knowledge or control knowledge to manage their enterprises. This situation needs to be changed, if we want to improve the situation of SMEs.

This simple ABC method was studied during a one year project. It’s not a long period, but it’s a beginning for future work – light-weight ABC in SMEs implementation in a tool. In this studied period, only a model for Technology for light-weight ABC in SMEs was developed, but its gives a theme and real situation in ABC usage. Initially only one industry was studied– milk manufacturing industry, where the insight was experimented with real financial data and researched standard processes of the possibilities in SMEs use in this business.

This research lets us consider non-commercial tools. Tools may be one of the key factors in making progress in SMEs businesses, since their use does not require extensive financial resources, usually they are free of charge and in current economical situation it’s a very important fact for L-W ABC implementation.

References


