Master Thesis in Software Engineering and Management

Software Performance Evaluation using UML-Ψ (PSI)

Omer Nauman Mirza
Göteborg, Sweden 2007

Department of Applied Information Technology
Software Performance Evaluation using UML-Ψ

Performance Engineering: Using UML-Ψ with SPE

OMER NAUMAN MIRZA
Software Performance Evaluation using UML-Ψ
Performance Engineering: Using UML-Ψ with SPE
OMER NAUMAN MIRZA

© OMER NAUMAN MIRZA, 2007

Report no 2007:118
ISSN: 1651-4769
Department of Applied Information Technology
IT University of Göteborg
Göteborg University and Chalmers University of Technology
P O Box 8718
SE – 402 75 Göteborg
Sweden
Telephone + 46 (0)31-772 4895

Göteborg, Sweden 2007
ACKNOWLEDGEMENTS

My last remaining task is to acknowledge all those people that have contributed to the work described in this thesis. This is an impossible task, given the many people that have helped to design, implement, apply, criticize, and evangelize the work. I am going to try anyway, and if your name is not listed, rest assured that my gratitude is not less than for those listed below.

At the outset, I would like to express my appreciation to Doctor Thomas Lundqvist for his advice during my master thesis research endeavor for the past 8 months. This thesis grew out of a series of dialogues with him. His observations and comments helped me to establish the overall direction of the research and to move forward.

I am also indebted to Professor Minhaj Ahmad Khan for suggesting me some ideas about the thesis. I am very grateful to Doctor Agneta Nilsson, whose guidance during the thesis preparation course paved my way in the field of scientific research.

Last, but not least, I wish to express my love and gratitude to all my family and friends. I would particularly like to thank my parents for their support during all these years.

Omer Nauman Mirza, Master Student
IT University of Göteborg
October 2007
Email: omer@ituniv.se
ABSTRACT

Predicting the performance of software architectures during early design stages is an active field of research in software engineering. It is expected that accurate predictions minimize the risk of performance problems in software systems. This would improve quality and save development time and cost of bug fixing in the later stages. SPE (Software Performance Engineering) and UML-Ψ (PSI) are among those methods which are used for performance prediction and helping the software architect to catch performance bottlenecks early.

SPE is a relatively mature approach and makes use of execution graphs and message sequence charts for the system’s representation. On the other hand, UML-Ψ uses Use Case, Activity and Deployment diagrams for the system’s representation. Use Case diagrams correspond to workloads applied to the system. Activity diagrams provide a high-level description of the computation steps performed by the system, and Deployment diagrams describe the physical resources on which the computations take place.

This thesis presents the research aimed at using UML-Ψ together with SPE. This is done by implementing a performance analysis example in these two methods. One of the contributions in the thesis is the definition of translation mechanism devised to facilitate the translation from SPE models and UML-Ψ models. However, only a subset of SPE models is considered during the thesis. Another contribution is done by identifying some improvements with respect to the uml modeling tools to increase the worth of UML-Ψ. The results presented in the thesis show how to use UML-Ψ with SPE by properly translating the models from one form to another.

Keywords: Software Performance, Performance Engineering, SPE, UML, UML-Ψ, Software Architecture.
# Table of Contents

1 INTRODUCTION .................................................................................................................. 1

1.1 OUTLINE OF THE REPORT ................................................................................................. 3

2 BACKGROUND ....................................................................................................................... 4

2.1 SOFTWARE PERFORMANCE ENGINEERING ...................................................................... 4
2.2 INTRODUCTION TO UML .................................................................................................. 5
2.3 UML-Ψ (PSI) ...................................................................................................................... 10

3 METHOD .............................................................................................................................. 13

3.1 RESEARCH APPROACH ..................................................................................................... 13
3.2 DESIGN RESEARCH .......................................................................................................... 13
3.3 DATA COLLECTION .......................................................................................................... 14
3.4 MOTIVATION FOR THE METHOD ...................................................................................... 14

4 RESULTS ............................................................................................................................. 15

4.1 TRANSLATING SPE MODELS INTO UML-Ψ MODELS .................................................. 15
4.2 ATM EXAMPLE IN UML-Ψ AND SPE .............................................................................. 16
4.3 TRANSLATION ISSUES ...................................................................................................... 28
4.4 TOOL MATURITY ISSUES ................................................................................................. 29

5 DISCUSSIONS ...................................................................................................................... 31

5.1 DISCUSSION OF THE RESULTS ....................................................................................... 31
5.2 DISCUSSION OF THE METHODOLOGY .......................................................................... 31
5.3 DISCUSSION FROM A SOFTWARE ARCHITECTURE PERSPECTIVE ................................ 32

6 CONCLUSIONS .................................................................................................................... 34

6.1 FUTURE WORK ................................................................................................................. 34

REFERENCES .......................................................................................................................... 36
1 Introduction

Performance is any characteristics of a software product that you could, in principle, measure by sitting at the computer with a stopwatch in your hand. The scope of performance includes responsiveness (response time or throughput) and scalability. “Performance is the degree to which a software system or components meets its objective for timeliness “[1]. Performance is an important quality attribute, but unfortunately it is given less attention during the development [1].

How do project get in trouble with performance? This problem is often raised because of a fundamental misunderstanding of how to achieve performance objectives. The approach is frequently “first let’s make it run; then, we will make it run fast”. The idea is to get the functionality right and then switch to performance. Unfortunately, by the time the architecture and design are selected, it may already be too late to achieve adequate performance by switching. It is a mistake to treat a potential performance problem as if it were of the same complexity as a coding error. Coding errors are not easy to fix, whereas performance problem may require extensive changes to system design. The proper way to manage software performance is to systematically plan for and predict the performance of the emerging software throughout the development process. And the best way to do this is to make appropriate decisions at the design time. Same approach is presented and used by C. Smith and L. Williams [1] in their proposed performance solutions.

Software performance engineering (SPE) [1, 5] provides a systematic, quantitative approach to construct software systems that meet performance objectives. With SPE, one can detect problems early in development, and use quantitative methods to support cost-benefit analysis of hardware solutions versus software requirements or design solutions, or a combination of software and hardware solutions [1]. The techniques presented in the book [1] by Smith and Williams, are collectively known as software performance engineering (SPE).

UML-Ψ [2] (also written as UML-PSI) is a software performance evaluation tool, which is based on a process-oriented simulation. Performance-annotated UML diagrams in the XMI file format [21] are the input for the UML-Ψ tool. The annotations are made according to the UML profile for schedulability, performance and time [18], and can be entered in regular UML modeling tools such as Poseidon [19] and ArgoUML [20]. They consist of arrival rates of requests, duration times for activities, and the speed of hardware resources etc. UML-Ψ automatically generates an event-based simulation from these UML-diagrams and executes it. The results of the simulation, like response times for scenarios or utilization of certain hardware devices, are reported back into special result attributes of the UML profile contained in the XMI-files. The developer may then re-import the diagrams into his UML-tool and directly identify performance problems within the architecture.

There are several problems with UML-Ψ such as: from the given UML-Ψ help, it is not very obvious that how UML-Ψ plants on different operating systems and there is a possibility of facing various problems in its compilation in windows environment due to
linking of library and software versioning issues (compatibility of old software). Moreover, Smith and Williams proposed a tool (i.e. SPE.ED [40]) to make use of SPE techniques. UML-Ψ is an open source [33, 34] and available under GNU General Public License [6]. However, SPE.ED is not open source. Open source [33, 34] availability of UML-Ψ is also a motivation for this study. This work will help to check the validity of UML-Ψ together with SPE techniques (used by the book examples [1]). In order to make SPE techniques work with UML-Ψ, one is required to implement simulation models according to the recommended practices of UML-Ψ.

This study aims to give the answer to the following research question:

*Is it possible to use UML-Ψ together with SPE (software performance engineering) and how can it be done?*

This research question was motivated by reading about the two methods. SPE is quite mature one and has a tool support in form of SPE.ED [40], which is not freely available. On the other hand, UML-Ψ is a new tool and is freely available. This particular research question helped in the evaluation of the UML-Ψ. A possible way to evaluate a tool can be: to start using that tool. Both positive and negative results to the research question are possible. It depends upon the results of simulation models. For positive reply, a successful simulation in UML-Ψ, as predicted in SPE book, will give the similar results. Certain factor (such as how fast it responds, throughput, scalability…) can be considered for the evaluation of success. In case of negative reply, it will provide reasons (i.e. why UML-Ψ cannot be used together with SPE due to errors in simulation models) in terms of errors and problems faced while trying UML-Ψ. It is also possible to come across some improvements in UML-Ψ as compared SPE, which is a matured method for performance evaluation. It will also help to highlight some of the differences between the two methods. Overall it provides an opportunity to check the two methods of performance evaluation.

In this study, it was chosen to experiment with UML modeling tools (i.e. both Poseidon V1.4 and ArgoUML V0.12 are recommended by UML-Ψ developers in [3]) to make use case, activity and deployment diagrams from given examples of the book [1]. Most of the work was done using ArgoUML. However, Poseidon V2.4 (i.e. a license version, which is available to It-university students) also used to check the consistency of the tag values between the two UML modeling tools. It is obligatory by the UML-Ψ developers [3] that the selected UML modeling tool must be capable of exporting the model in XMI format [9]. So far, UML-Ψ only accepts XMI format [3, 9]. In this work, ArgoUML is opted due to its open source nature and relatively simple configuration with Ubuntu [22]. UML-Ψ considers following UML diagrams to derive the performance model: Use Case, Activity and Deployment diagrams. Then with UML-Ψ, a process oriented simulation model of the software system was built. The simulation program was finally executed and computed a set of performance indices of the software system under study: resources utilization and throughput, and the mean execution time of actions and Use Cases. Simulation results were reported back into the original software model as UML tagged values associated to the relevant elements. This provided a user-friendly feedback at the
software design level [3]. Figure 1 illustrates an overall schematic representation of UML-Ψ usage.

Similar work has been done by Balsamo et. al. [15], who compares an analytical and a simulation based method in a case study. Yet they do not compare the prediction of the methods with measurements from an implementation. Such a comparison of prediction and measurement has been done by Gorton et. al. [16] for a software architecture based on Enterprise Java Beans, but only with a single method. Another work is done by Koziolek and Firus et. al.[14]. They evaluated three methods including both UML-Ψ and SPE using GQM [17] approach. The third method, which they evaluated, is capacity planning (CP) [14].

1.1 Outline of the report

The following chapter, Background, gives an introduction of what SPE and UML-Ψ are, and presents relevant background including UML diagrams (i.e. use-cases, activity and deployment diagrams). The Method (Chapter 3) describes how this study was conducted. The Results (Chapter 4) presents the results of the conducted study while the Discussion (Chapter 5) analyzes the results and method, and it also provides an overall discussion about the software performance from the software architecture perspective. In the Conclusion (Chapter 6) the most important outcomes of the study are presented and in the Future work the recommendations for future studies are suggested.
2 Background

This chapter presents the background to the study and gives an overview of Software Performance Engineering (SPE), UML (i.e. use cases, activity and deployment diagram) and UML-Ψ. The UML section is restricted to use cases, activity and deployment diagrams. This is done in compliance with UML-Ψ usage [24]. More information about UML can be found in [23, 25 26].

2.1 Software Performance Engineering

Performance is an important quality attribute of software architecture. It can be characterized by metrics such as response time, throughput, and resource utilization [1]. In many existing systems, the reason for bad performance is a poorly designed software architecture [1]. Performance predictions based on architectural descriptions of a software system can be performed before the implementation starts, which can possibly reduce cost for subsequent changes to fix performance problems. It is the hope that such early analyses support the decision for design alternatives and reduce the risk of having to redesign the architecture after performance problems have been diagnosed in the implementation.

Software performance engineering (SPE) [1, 31] provides a systematic, quantitative approach to constructing software systems that meet performance objectives. The “performance balance” in Figure 2 depicts a system that fails to meet performance objectives because its resource requirement exceeds computer and network capacity. With SPE, one can detect problems early in development, and use quantitative methods to support cost-benefit analysis of hardware solutions versus software requirements or design solutions, or a combination of software and hardware solutions.
SPE is a software-oriented approach; it focuses on architecture, design, and implementation choices. It uses model predictions to evaluate trade-offs in software functions, hardware size, quality of results, and resource requirements. Following are some modeling strategies, being used by the SPE: The simple model strategy \cite[p.18]{1}, The best- and worst-case strategy \cite[p.19]{1} and The adapt to precision strategy \cite[p.19]{1}. SPE uses these strategies to obtain results quickly, to cope with uncertainty in estimates of software and hardware resource usage, and to control costs. The models assist developers in controlling resource requirements by enabling them to select architecture and design alternatives with acceptable performance characteristics. The models aid in tracking performance throughout the development process and prevent problems from surfacing late in the life cycle (typically during final testing).

SPE also prescribes principles and performance patterns for creating responsive software, performance anti-patterns for recognizing and correcting common problems, the data required for evaluation, procedures for obtaining performance specifications, and guidelines for the types of evaluation to be conducted at each development stage. It incorporates models for representing and predicting performance as well as a set of analysis methods.

\subsection{Introduction to UML}

UML is a semi formal language developed by the OMG \cite{27} for specifying, visualizing and documenting software artifacts; it can also be applied to non-software systems, such
as business processes. UML is widely used in the software engineering community to describe systems developed according to the object-oriented paradigm.

UML is a graphical notation which allows the user to describe an artifact using a suitable combination of diagrams, chosen among the available ones. UML defines several kinds of diagrams:

- Use Case diagram;
- Class diagram;
- Behavior diagrams: Statechart diagram, Activity diagram;
- Interaction diagrams: Sequence diagram, Collaboration diagram;
- Implementation diagrams: Component diagram, Deployment diagram

The above diagrams can be partitioned in three categories:

- Static diagrams are used to model the logical or physical structure of the system. They include Class diagram, Component diagram and Deployment diagram.
- Dynamic diagrams are used to describe the behavior of the system. They include Use Case diagram, Statechart diagram, Activity diagram, Sequence diagram, Collaboration diagram.
- Model Management diagrams are used to group other model elements, and include Component diagram.

2.2.1 Use Case diagram

Use case diagrams describe at a high level the interaction between the system and actors requiring service. An actor is any entity (physical and logical) which may interact with the system. Actors are graphically represented as “stick people”; they can interact with the system in possibly different ways, each being a different use case. Each use case represents one or more scenarios. A use case is graphically represented as an oval connected to an actor. This connection may represent the fact that the actor generates or takes part to the use case.

UML defines different kind of relations between use cases:

extend: An extend relationship between use cases is shown by a dashed arrow with an open arrow-head from the use case providing the extension to the base use case. The arrow is labeled with the keyword <<extend>>. An extend relationship from use case A to use case B indicates that B may be augmented by the additional behaviors specified by A. The user may indicate explicitly the extension points inside the use case A.
**include**: An include relationship between use cases is shown by a dashed arrow with an open arrow-head from the base use case to the included use case. The arrow is labeled with the keyword `<<include>>`. An include relationship from use case A to use case B indicates that A will also contain the behavior specified by B.

![Include Relationship Diagram]

Generalization: A generalization between use cases is shown by a generalization arrow, that is, a solid line with a closed, hollow arrow head pointing at the parent use case. A generalization from use case A to use case B indicates that A is a specialization of B.

![Generalization Diagram]

### 2.2.2 Activity diagram

An activity diagram is a special case of a state diagram in which most of the states are sub-activity states or action states (states corresponding to the execution of an action). Moreover, transitions in an activity diagram are generally triggered by completion of the actions or sub-activities in the source states. An activity diagram specifies the behavior of a use case, a package, or the implementation of an operation.

Activity diagrams are an evolution of flow charts, from which they inherit the ability to show execution flows depending on internal processing (as opposed to external events). State diagrams are preferred in situations where external, asynchronous events occur. Activity diagrams include the possibility to represent concurrent execution of multiple computations through fork/join nodes. A fork node denotes the point where the computation splits in concurrent execution threads, each evolving independently from the others. A join node denotes a point where different execution threads synchronize.
2.2.3 Deployment diagram

Deployment diagrams (see Figure 4) show the configuration of run-time processing elements and the software components, processes, and objects that execute on them. Software component instances represent run-time manifestations of software code units. Components that do not exist as run-time entities (because they have been compiled away) do not appear on these diagrams, but should be shown on component diagrams. A deployment diagram is a graph of nodes connected by communication associations. Nodes may contain component instances: this indicates that the component runs or executes on the node. Components may contain instances of classifiers, which indicate that the instance resides on the component. Components are connected to other components by dashed-arrow dependencies (possibly through interfaces). This indicates that one component uses the services of another component. A stereotype may be used to indicate the precise dependency, if needed. The deployment type diagram may also be used to show which components may reside on which nodes, by using dashed arrows.
with the stereotype \texttt{<<deploy>>} (see section 2.2.4.1 for stereotypes) from the component symbol to the node symbol or by graphically nesting the component symbol within the node symbol.

![Figure 4: A UML deployment diagram.](image)

### 2.2.4 Extension Mechanisms

UML provides some built-in functionalities for extending its metamodel. Such extension mechanisms can be used to add specific informations to existing UML elements, or to define new types of metamodel elements based on existing ones. The constraint on all extensions defined using the extension mechanism is that extensions must not contradict or conflict the standard semantics. Thus, extensions must be strictly additive to the standard UML semantics. The extension mechanisms are a means for refining the standard semantics of UML, but do not support arbitrary semantic extension [26]. UML extension mechanisms include Stereotypes and Tagged Values.

#### 2.2.4.1 Stereotypes

Stereotypes are the main UML extension mechanism. They are used to define subclasses of existing metamodel elements. Such new metamodel elements have the same structure as the original one, but may have additional constraints from the base metamodel class, or it may require tagged values (see below) to be included to the elements with that stereotype.

A stereotype is graphically represented as a textual label enclosed in double quotes (eg \texttt{<< abstract>>}). The label is associated to the model element to be stereotyped.

#### 2.2.4.2 Tagged Values

Tag definitions specify new kinds of properties that may be attached to model elements. The actual properties of individual model elements are specified using Tagged Values. These may either be simple datatype values or references to other model elements.
Tagged values may be used to represent properties such as code generation information or quantitative information for performance evaluation purpose.

### 2.2.4.3 UML Profiles

A profile is a stereotyped package that contains model elements that have been customized for a specific domain or purpose by extending the metamodel using stereotypes, tagged definitions, and constraints. A profile may specify model libraries on which it depends and the metamodel subset that it extends. Examples of UML profiles include the UML profile for Schedulability, Performance and Time specification [28].

### 2.3 UML-Ψ (PSI)

UML-Ψ is a software performance evaluation tool which uses UML for software specification, and process oriented simulation as the performance model. The approach integrates UML software specification given by a set of annotated diagrams (use case, activity and deployment diagrams), with a discrete-event simulation model whose solution gives a set of average performance indices providing automatic feedback at the software architectural level. The UML-Ψ tool can be used by both the software designer [24] and the performance modeler [24] to generate and evaluate the performance model.

Figure 5 illustrates the high-level structure of the UML-Ψ model processing framework. The UML model is created using a UML modeling tool, and is exported in XMI format. XMI is an XML-based notation for describing UML models, useful for exchanging UML models among different applications. Due to incompatibilities among different implementations of the XMI format, UML-Ψ currently supports the XMI dialect used by the open-source ArgoUML [20] tool version 0.12, and its commercial counterpart Poseidon [19] version 1.4. Supporting other UML modeling tools is possible, by simply implementing an appropriate XMI parser module.
The UML-Ψ builds a process-oriented simulation model [30] from UML specifications (i.e. by parsing Use Case, Activity and Deployment diagrams). The performance model is composed by a set of concurrent, interacting simulation processes. Authors [2] define three types of simulation processes: workloads, actions and resources. Workload processes generate sequences of requests to the system; workloads may be open or closed. An open workload represents an infinite stream of requests being generated from outside the system, while a closed workload is made of a fixed number of requests circulating through the system. Workloads are represented in use cases. Each request, upon arrival, triggers the execution of a sequence of actions. Sequences of actions are represented in activity diagrams. An action is a request of service from an active resource (e.g., processor) or acquisition/release of a passive resource (e.g., memory). Resources are highlighted by deployment diagram.

The UML model (i.e. Use Case, Activity and Deployment diagram) has to be annotated according to a subset of the Profile for Schedulability, Performance and Time Specification [28] as described in [2, 8]. Annotations are inserted into the UML model as stereotypes and tagged values; such annotations must be provided by the user before the performance model generation phase.

The UML-Ψ tool executes the simulation model by using both user-supplied parameters, that are given as tagged values associated to UML elements, and the parameters included in a configuration file. A complete description of the available tag names and values is available in [2]. UML-Ψ considers the specification of tag values by the Tag Value
Language (TVL), a subset of the Perl language [29] proposed in [28]. This is motivated by the need to express such values in a complex way, for example by using expressions such as arithmetic or boolean ones. In UML-Ψ, a configuration file is a user-defined Perl program. UML-Ψ parses the program and uses the Perl interpreter environment (modified by every declaration contained in the configuration file) to parse tag values. If a tag value contains an expression such as:

```
PAdemand = ['assm','dist','exponential', $mean]
```

the configuration file must specify a value for $mean, such as for example:

```
$mean=5.0;
```

The simulation performance model is eventually executed and the computed results are inserted into the XMI document as tagged values associated with the UML elements they refer to. Such results are available to the user which can open again the UML model by using the UML modeling tools (i.e. ArgoUML or Poseidon). The performance modeling cycle illustrated in Figure 5 may be iterated to meet given performance requirements.
3 Method
This chapter contains information about how the research approach was planned and performed. The information is based on the discussion of method for doing this research project.

3.1 Research Approach
The design research [11] approach was used as the method to answer the research question. This was done to satisfy the need of devising some simulation models through UML-Ψ of SPE book examples [1]. UML diagrams (i.e. use case, activity and deployment diagram) were designed from such UML modeling tools (i.e. ArgoUML v.12 [20]) which was capable of performing the diagrams conversion to XMI [21] format. Main focus was given to the evaluation of UML-Ψ. One of the possible outputs of the design research can be models as specified by March and Smith [13]. A proposed system (i.e. ATM example) from the book [1] was selected. In this work, simulation models were generated from UML-Ψ through which performance of the proposed system can be predicted (by calculating responsiveness, throughput, scalability…).

3.2 Design Research
In this study, (as said earlier) design research approach was adopted. It was helpful in UML-Ψ evaluation through simulation models, when it came to the evaluation stage. Figure 6 describes the design research process [12] and its various phases:

Figure 6: Design research process [11]

“Design research is sometimes called ‘Improvement Research’ and this designation emphasizes the problem-solving/performance-improving nature of the activity” [11]. From the design research process, Awareness of the problem is the consistency between implementation artifacts (i.e. uml diagrams) and analysis/design artifacts (i.e. uml
diagrams). Suggestion is SPE techniques together with UML-Ψ. Development is the simulation models (i.e. examples from the SPE book [1]) using SPE techniques that are evaluated with a view to modeling and implementing simulations with UML-Ψ. In this scenario, evaluation is the most important step for design research. In evaluation stage, simulation models (being done in UML-Ψ) through performance dimensions (i.e. responsiveness, scalability …) [1] were evaluated. Conclusion is the end of the project [11].

3.3 Data Collection

Data collection was done by software simulation development diary, which edited on daily basis. Initially, the data was collected from compilation errors of the UML-Ψ. When it came to compilation of libcpsim [6] (i.e. a C++ libraries used by UML-Ψ), it was expected to get different response in different operating system. The artifacts (i.e. use case, activity and deployment diagrams) were produced for simulation and used by UML-Ψ. Those artifacts were also compared with SPE diagrams for consistency. Besides this, scenarios were performed (e.g. change of a use case) and then throughput (i.e. a way to measure performance [4]) for both SPE and UML-Ψ simulation was recorded. In software simulation development diary, the results of simulation models with respect to UML diagrams (i.e. from both UML-Ψ examples and SPE book examples) were added. Finally, for both UML-Ψ and SPE various notes and experiences (i.e. with respect to installation of UML-Ψ) are an important part of the conclusions section. Error notifications from the compilation of both library (i.e. libcpsim) and UML-Ψ are also an integral part of the results section.

3.4 Motivation for the method

The motivation for this study and the method was that the research question and research area have not yet been studied or compared exactly in a published study. The method used were considered to be able to give sufficient results in order to answer the research question of this study, by implementing some simple examples from SPE into UML-Ψ. The decision to have some simple example in the beginning was due to the fact that it would be more likely to face some problems in the beginning with the configuration of the tool. One has to define some translation mechanism from SPE models to UML-Ψ models. And due to the nature of research question, design research considered as a natural choice.
4 Results

The Background chapter informed about UML-Ψ and SPE. This chapter presents the results obtained from this study. Section 4.1 informs about how translation was done from SPE to UML-Ψ. Section 4.2 provides an example implemented with the two methods (i.e. SPE and UML-Ψ). Section 4.3 highlights the problems faced during the translation. Section 4.4 discusses some issues about the UML-Ψ maturity, which came across during it’s evaluation.

4.1 Translating SPE models into UML-Ψ models

SPE models differ from UML-Ψ models in many respects. These differences are highlighted in this section of the report. Some conversion is required to use SPE models into UML-Ψ models. And it provides one possible way of translating SPE models into UML-Ψ models.

Transforming SPE models into UML-Ψ models can done by considering the requirements for both the models (i.e. SPE models use execution graphs and message sequence charts (MSC). On the other hand, UML-Ψ uses use cases, activity and deployment diagrams to model the architecture of software system). In SPE models, Message sequence charts are drawn to fulfill the use of scenarios. From these charts, execution graph are drawn. By following the guidelines [1, pp.72-88], execution graphs are drawn from sequence charts. In the book [1], authors have made use MSCs. However, MSCs can be replaced by activity diagrams. The translation from activity diagrams to execution graphs will be difficult [1, P89]. In that case, branch symbol could describe either repetition or alternation and one has to define stereotypes for repetition and alteration to overcome this issue. Figure 7 depicts some basic execution graph notations.
UML-Ψ (as described in background section) makes use of use cases, activity and deployment diagram for performance evaluation. For performance evaluation using UML-Ψ, it is necessary to convert these notations into these diagrams. Scenarios are represented in the use cases. Basic nodes are represented in activity diagrams. Case node is not drawn in activity diagram. In UML-Ψ, each activity diagram is drawn against a specific scenario. So number of activity diagrams will be equivalent to the number of use cases. Expanded node is not possible to depict in UML-Ψ. This limitation restricts the use of multi-step process. So instead of multi-step, every step is mentioned in terms of basic nodes. This will be cleared in the later section of the report with ATM example (i.e. instead of one multi-process step for withdraw, deposit and balance inquiry, a separate activity diagram is drawn for each use case in UML-Ψ. So, a multi-step node is divided into small steps.). Pardo Node is handled with join in the activity diagram. However, alternate for Split node is not yet explored in UML-Ψ.

### 4.2 ATM Example in UML-Ψ and SPE

To illustrate the difference between SPE and UML-Ψ for modeling and evaluating the performance of software systems, consider an example which is based on a simple automated teller machine (ATM).
The ATM accepts a bank card and requests a personal identification number (PIN) for user authentication. Customers can perform any of three transactions at the ATM: deposit cash to an account, withdraw cash from an account, or request the available balance in an account. A customer may perform several transactions during a single ATM session. The ATM communicates with a computer at the host bank which verifies the account and processes the transaction. When the customer is finished using the ATM, a receipt is printed for all transactions and the customer’s card is returned.

4.2.1 ATM example in SPE

This SPE example is compiled and edited from the book [1] examples and a paper about SPE.ED (a tool which is built on making use of SPE approach) [37].

Here, the authors [1] focus on scenarios that describe the use of the ATM. A full specification would include additional models, such as a class diagram and behavior descriptions for each class. However, they omitted additional models due to their primary interest in the use of scenarios as a bridge between Object-Oriented Development and Software Performance Engineering.

As described in [35], scenarios represent a common point of departure between object-oriented requirements or design models and SPE models. Scenarios may be represented in a variety of ways [36]. Here, we use Message Sequence Charts (MSCs) to describe scenarios in object-oriented models.

Figure 8 illustrates a high-level MSC for the ATM example. Each object that participates in the scenario is represented by a vertical line or axis. The axis is labeled with the object name (e.g., anATM). The vertical axis represents relative time which increases from top to bottom; an axis does not include an absolute time scale. Interactions between objects (events or operation invocations) are represented by horizontal arrows.

Figure 8 describes a general scenario for user interaction with the ATM. The rectangular areas labeled “loop” and “alt” are known as “inline expressions” and denote repetition and alternation. This Message Sequence Chart indicates that the user may repeatedly select a transaction which may be a deposit, a withdrawal, or a balance inquiry. The rounded rectangles are “MSC references” which refer to other MSCs. The use of MSC references allows horizontal expansion of Message Sequence Charts. The MSC that corresponds to Process Withdrawal is shown in Figure 9.
Figure 8: Message Sequence Chart for User Interaction with the ATM [1]
A Message Sequence Chart may also be decomposed vertically, i.e., a refining MSC may be attached to an instance axis. Figure 10 shows a part of the decomposition of the \texttt{anATM} instance axis. The dashed arrows represent object instance creation or destruction.

### 4.2.1.1 Mapping Scenarios to Performance Models

Models for evaluating the performance characteristics of the proposed ATM system are based on performance scenarios for the major uses of the system. These performance scenarios are the same as the functional scenarios illustrated in the message sequence charts (Figures 8 to 10). However, they are represented using Execution Graphs. Note that not all functional scenarios are necessarily significant from a performance perspective. Thus, an SPE study would only model those scenarios that represent user tasks or events that are significant to the performance of the system. The selections of such scenarios are done by a performance walkthrough [1].
Figure 10: Partial Decomposition of an ATM [1]
Figure 11: Execution graph from ATM scenarios [1]

Figure 11 shows an Execution Graph illustrating the general ATM scenario. Various execution graph notations are available in [1, p77]. In figure 11, the case node indicates a choice of transactions while the repetition node indicates that a session may consist of multiple transactions. Subgraphs corresponding to the expanded nodes show additional processing details. The processing steps (basic nodes) correspond to steps in the lowest-level Message Sequence Chart diagram for the scenario. The execution graph in Figure 11 shows an end-to-end session that spans several ATM customer interactions. Thus analysts can evaluate the performance for each individual customer interaction as well as the total time to complete a session.

4.2.1.2 Model Solution and Results

The analyst first solves a ‘No Contention’ model to confirm that in the best case, a single ATM session will complete in the desired time, without causing performance bottlenecks at the host bank. Up to four sets of results may be displayed concurrently, as shown in Figure 12.

The elapsed time result for the ‘No Contention’ model is in the top-left quadrant. The overall time is at the top, and the time for each processing step is next to the step. The color bar legend in the upper right corner of the quadrant shows the values associated with each color; the upper bound is set by defining an overall performance objective. Values higher than the performance objective will be red, lower values are respectively cooler colors. The ‘Resource usage’ values below the color bar legend show the time spent at each computer device. Of the 35.6 total seconds for the end-to-end scenario, 35.25 is due to the delays at the ATM unit for customer interactions and processing. Thus, no performance problems are apparent with this result.

The SPE.ED [37, 40] tool evaluates the results of device contention delays by automatically creating and solving an analytic queuing network model. The utilization
result for the ‘Contention solution’ of the ATM sessions with an arrival rate of 5 withdrawal transactions per second is in the top-right quadrant of Figure 7. The total utilization of each server is shown under the color bar, and the utilization of each device by each processing step is next to the step. The total CPU utilization is 15%, and the disk device is 100%. Even though the customer database would fit on one disk device, more are needed to relieve the contention delays. In general, options for correcting bottlenecks are to reduce the number of I/Os to the disk, reduce the number of ATM units that share a host bank server, or add disks to the server. The options are evaluated by changing software processing steps, or values in the overhead matrix.

Figure 12: SPE model results [1] using SPE.ED
The results in the lower quadrants of Figure 12 show the utilization and response time results for 3 disk devices. The quadrants let the analyst easily compare performance metrics for alternatives.

### 4.2.2 ATM example in UML-Ψ

As it is written earlier in section 2.3, UML-Ψ makes use of three diagrams for performance evaluation (i.e. Use cases, Activity and Deployment Diagram). For the ATM example, these diagrams are drawn together with their respective tag values and stereotypes. The different scenarios (deposit, withdrawal and balance inquiry) are drawn in terms of use cases. Then for each scenario an activity diagram is drawn. In SPE it is possible to draw all the three scenarios with the help of a case node [1, p.77] in the execution graph. The same is not possible to do in an activity diagram with UML-Ψ. UML-Ψ has difficulties in handling branches. And due to versioning issues (see sections 4.3 and 4.4 for details), for each scenario a different activity diagram is drawn.

Figure 13 depicts the use case diagram for the ATM example.

![ATM use case diagram](image)

**Figure 13: ATM use case diagram**

With UML-Ψ, it is not possible to use multi state(s) or pseudo state(s) (as one can do in SPE). So, initially it is required to draw a use case diagram with processDeposit, processWithdrawal and processBalanceInquiry use cases. Figure 13 depicts the use case diagram with a user. In a use case diagram, actors represent workloads, and use cases represent scenarios which can be activated by the requests. Two different stereotypes, <<OpenWorkload>> and <<ClosedWorkload>>, are defined to denote open and closed workloads respectively. These stereotypes are associated with actors. In ATM example, user is stereotyped with OpenWorkload stereotype to make the number of requests unlimited [2].

---

1 This diagram is captured from MSVisio to increase readability (same is done with other UML diagrams.). In ArgoUML [20] V0.12, it is not possible to show tag values and stereotypes. Tag values and stereotypes are visible only in a property box, when a specific use case or activity is selected.
Figure 14: Activity Diagram for ATM [1, P59]

Figure 14 depicts the general activity diagram which handles all the three scenarios. It is included to illustrate some issues with UML-Ψ. However, three activity diagrams (From diagram 15 to 17) are also drawn, which represent one scenario from the use case diagram. In this example, all action states are stereotyped As <<PAStep>> [2], and thus represent computations performed on some resource, whose name is specified in PAhost tag value. Those names correspond to node names in the deployment diagram (see figure 18). All the three resources (i.e. ATM, CPU, Dev) are considered as active resources. This helps to add their PAschedPolicy [2] tag values.
Figure 15: Activity diagram for process deposit.
Figure 16: Activity diagram for process withdrawal.
Figure 17: Activity diagram for process balance inquiry.
Figure 18: Deployment diagram for ATM example.

Simulation results are summarized in Table 1. The results show that the utilization of CPU and DEV are quite low (15% and 53% respectively). However, for ATM it is quite large. And it is due to user, spending a lot of time in interaction with ATM as compared to CPU processing and DEV inquiry. Time needed to process deposit (about 30s) is higher than both withdrawal and balance inquiry. Results from both UML-Ψ and SPE are consistent.

<table>
<thead>
<tr>
<th></th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU utilization</td>
<td>0.1537</td>
</tr>
<tr>
<td>ATM utilization</td>
<td>0.8839</td>
</tr>
<tr>
<td>DEV utilization</td>
<td>0.5364</td>
</tr>
<tr>
<td>Process deposit response time</td>
<td>35.97</td>
</tr>
<tr>
<td>Process withdrawal response time</td>
<td>30.90</td>
</tr>
<tr>
<td>Process balance inquiry response time</td>
<td>15.94</td>
</tr>
</tbody>
</table>

Table 1: Simulation results using UML-Ψ for ATM example.

4.3 Translation issues

Translation from SPE models into UML-Ψ models has some problems to address. Some of these problems were found during the ATM example implementation using UML-Ψ. Others were originated from the use of ArgoUML V.12 [20]. These problems are presented in this section of the report.

A very first problem which one can face (came across during the ATM example) is probably the representation of a composite state. In SPE, one can make use of an expanded node (see figure 7) to represent composite states. Same is not true with UML-Ψ, due to the use of activity diagram in UML-Ψ instead of execution graphs. For each use case, one activity diagram is drawn together with tag values for each state. With argoUML [20] there is no possibility to draw a complex state in the activity diagram. In

---

1 Throughput index for CPU, ATM and DEV can also be calculated using UML-Ψ from PAthroughput tag. This tag is available for both PAhost (in case of active resource) and PAresource (in case of passive resource) stereotypes.
the activity diagram, every possible step must be represented separately from the other step. So, it seems that SPE is a bit more high level, where one can make use of complex states to skip some steps.

Another translational problem arises when one is looking to draw a case node (see Figure 7). Same motivation is true for this as given for composite state. The activity diagrams are drawn from the use cases and for one use case there is one corresponding activity diagram. With case node in SPE, one can choose from different transitions together with the probability of each one to occur (see ATM example in Figure 11, where process deposit, process withdrawal and process balance inquiry are drawn with the help of a case node. In UML-Ψ, for each scenario one has to draw a separate activity diagram.

The representation of repetition of a step in UML-Ψ from execution graph is handled differently than in SPE. Instead of drawing a loop in the activity diagram, a PArep tag is used together with its value (i.e. number of times it should be repeated).

This work didn’t look at software resources as used in SPE. And the ATM example in SPE hardly used any software resource. So it is not possible to give any results on how software resources should be translated.

### 4.4 Tool maturity issues

This section highlights some important issues, which were originated due to some software constraints (i.e. UML modeling tools, Perl and C compiler) for UML-Ψ.

Using UML-Ψ for performance evaluation requires some technical tasks to take care, before actually building the UML diagrams (i.e. use cases, activity and deployment diagrams). UML-Ψ requires the UML model to be exported in XMI format. So far UML-Ψ understands the XMI dialect produced by the freely available ArgoUML modeling tool [20], and its commercial counterpart Poseidon [80]. From these two, only one versions of each tool is tested with UML-Ψ (i.e. ArgoUML version 0.12 and Poseidon version 1.4). UML-Ψ was also tested with ArgoUML Version 0.24 (the latest one) and Poseidon Version 2.4.1. But UML-Ψ is known not to work with Poseidon version 2.4.1, as the XMI format it employs is different and incompatible with previous versions. Same is the case with Argo UML Version 0.24. Even if one tries to open UML models made in the older versions of the two UML modeling tools (i.e. ArgoUML and Poseidon), it skips tag values in activity diagrams and base classes [2] for stereotypes are changed (i.e. from their original base class to ModelElement as a general base class).

UML-Ψ was tested on two different platforms (i.e. Windows XP and Ubuntu6.10). In Windows XP, UWIN [39] was also installed together 8 different setup files, which are as follows:

- **uwin-base** - Which contains the UWIN runtime, korn shell, daemons, services and over 100 command line tools you find in Unix/Linux.
- **uwin-dev** - Contains the compiler 'cc', make, as well as the libraries needed to compile the Unix applications for running natively on Windows.
- **uwin-groff**
• **uwin-perl** - Perl language packages so you can run perl programs.
• **uwin-terminfo**
• **uwin-xbase** - X windows base packages needed to run X applications (provided you have an X server running).
• **uwin-xdev** - X windows libraries needed to develop X applications.
• **uwin-xfonts** - All necessary fonts for X windows.

Out of these 8, it is required to have at least 4 setup files (i.e. uwin-base, uwin-dev, uwin-perl, uwin-xfonts) for UML-Ψ compilation. It requires perl interpreter to interpret the script written in perl for variables in tag values. This script contains the variables together with their respective values. These variables are then used in tag values with UML-Ψ. The basic simulation entity provided by the library is the simulation process: methods are provided for activating, stopping and rescheduling simulation processes. The library provides simulation primitives which are commonly implemented in many process oriented simulation languages or libraries, such as the Simula language [38]. However, when UML-Ψ was compiled with UWIN, it couldn’t compile due errors with its compiler ‘cc’ and make. Compiler generated errors when it tried to compile libcppsim. Libcppsim [2, P95] is a process oriented simulation library written in C++. Same versions for gcc and perl were used as prescribed by the UML-Ψ developers. Errors occurred in the first phase, when ‘./configure’ and ‘make’ commands were issued over the prompt to configure libcppsim.

Due to compilation errors with UWIN and XP, UML-Ψ was compiled with Ubuntu 6.10. This time no compilation error was found. The C compiler used, is gcc 4:4.1.1-6ubuntu3; it was also confirmed that UML-Ψ (and libcppsim) is known not to compile correctly under gcc 3.x, due to differences in the Standard Template Library. In order to make UML-Ψ work together with latest Linux distributions, this issue must be addressed. UML-Ψ requires Perl to be installed in the system, as it must be linked against the Perl interpreter in order to parse tag values. The version of Perl shipped with Ubuntu 6.10 is v5.8.8-6. Later given example was checked for its working by setting the path for libcppsim and source directory of UML-Ψ (as mentioned in the install file for UML-Ψ). After printing some debug messages, simulation was started. Here simulator also checks for convergence point to reach to stop the simulation. One can also specify maximum simulation time to avoid the wait for convergence. Then it will print the simulation results after that specified point of time, irrespective to whether convergence has occurred or not.
5 Discussions

In this chapter the discussion of the study is presented. Section 5.1 discusses the result of the study, followed by a discussion of the method in section 5.2. Finally, in section 5.3, performance evaluation in general is discussed from the Software Architecture.

5.1 Discussion of the Results

The results show that it is quite possible to use UML-$\Psi$ together with SPE. But at the same time, modifications are required in the SPE models to facilitate a smooth transformation. UML-$\Psi$ accepts only use-case, activity and deployment diagrams and is therefore bound to the modeling possibilities of these three diagram types. So SPE models are first required to be converted into these diagrams. Execution graphs are converted to activity diagrams. Then in SPE there is a possibility of having message sequence chart. It would be really nice if UML-$\Psi$ would have supported sequence diagram or state chart diagram together with activity diagrams. State charts put more emphasis on the notion of “event”, while activity diagrams put more emphasis on the notion of “activity” taking some amount of time to complete before starting the next one. Addition of sequence diagram might be helpful with software resources.

The UML-$\Psi$ is suggested to go through some modification in terms of it’s used libraries (i.e. libcpsim) and supported UML modeling tools. Such modifications would not only increase its usability but also ameliorate it’s worth. As the results indicate, currently it supports only two UML modeling tools with their specific versions. It can only read XMI files generated from either ArgoUML version 0.12 or Poseidon version 1.4. It would be better if it could at support the latest versions of both ArgoUML and Poseidon.

One can face a lot of problems while installing UML-$\Psi$, if one is not used to the linux platform. It becomes equally hard to catch different errors, which can occur during the compilation of libcpsim. The results also verify the findings by H. Koziolek and V. Firus [14] that UML-$\Psi$ usage is prone to errors especially in the beginning. There is a need for an installation guide together with a manual to exclude the extra burden for the user. Current installation help is not enough for the starters.

Currently UML-$\Psi$ comes with various files and a library to install. Then one has to model the architecture in the required case-tools and generate XMI file from the UML modeling tool. That XMI works as a input for UML-$\Psi$. It would be practical if UML-$\Psi$ was available as a plug-in for a modeling tool and had not to be started externally.

5.2 Discussion of the Methodology

This study started with the aim of learning about the two performance evaluation methods (i.e. SPE and UML-$\Psi$). One of main motive was to evaluate UML-$\Psi$ as a performance evaluation tool. It is relatively easy to measure if you have something to compare with or
match on. In this case, SPE is a well renowned approach. Therefore, it was tried to practice SPE models in term of UML-Ψ models. It produced interesting results in terms of models translation and tool evaluation.

As far as published papers and reports are concerned, no one may be implemented SPE examples in UML-Ψ. Therefore, a translation procedure was required to enable this. This was achieved through design research. In the beginning, it was planned to modify the UML-Ψ so that it can get input in some other format besides XMI. But defining a suitable translational mechanism and the UML modeling tool versioning issues took a lot of time.

The simulation development diary helped a lot during the installation of UML-Ψ and in comparison of results between the two methods. In the installation, it helped to keep track of various errors. For the results, it helped in the manual matching of the simulation results.

In this work, the original developer of UML-Ψ was not contacted for help about his tool. Its importance increases, when no user manual is available for the tool. Some times it was really hard to do the translation. When his given example was tried, it failed to work. It took a lot of time to recreate that example. It would have been better, if one example was implemented under his guidance as a training session. This could have improved results (in terms of number of examples implemented) and speed of obtaining results could have been much better.

The study is repeatable and the same research question can be used in a replication study to investigate if the result can vary with a person having more idea about performance engineering. It is quite possible for someone to come up with another translation mechanism. But in that case too, simulation results shouldn’t change a lot.

5.3 Discussion from a Software Architecture Perspective

Performance is one of software quality attributes from the perspective of software architecture [7]. Synthesis and analysis [7] are the two methods, used by software architects or performance engineers, for achieving performance. In large projects, a fulltime performance engineer will is advised [1]. However, in small projects software architect has the responsibility for maintaining the performance while designing the architecture.

Synthesis: methods used to synthesize (such as real-time design methodologies) a system or Smith’s software performance engineering philosophy as discussed in [5],[7]

Analysis: techniques used to analyze system performance such as queuing analysis and scheduling analysis. [7]

A correct performance related decision at the design time can reduce the cost and effort required to achieve performance in the later stages of development. Both UML-Ψ and SPE helps the software architect or performance engineer to reduce the performance bottlenecks. UML-Ψ might be most convenient for developers because it relies on UML
models (use case, activity and deployment diagrams) and does an automatic transformation of software models to performance simulations. It also report back results to XMI file, which can be opened in the same UML modeling tool to alter the design (if needed).

SPE is considered as a well practiced approach all over the performance engineering community. At the same time, UML-Ψ is a bit new to the horizon. Both SPE and UML-Ψ use different models for performance evaluation (see the background chapter). However, both SPE and UML-Ψ suggests changes (if required) in the design to achieve the performance objectives [1, p.172].
6 Conclusions

At a general level, this thesis is about introducing performance engineering and some of the current standards used by the industry for performance evaluation of the software systems. After reading about performance as an important quality attribute not only for critical software system but also for other applications, it can be concluded that there is a need to avoid the ‘fix it later’ approach. It seems that by omitting early performance analysis, one can still build software. But any late fixes of performance problems cost a lot more than if it would have been addressed in the design stage of the development [1].

The main objective of the research presented in the master thesis was to investigate the possibility of using UML-$\Psi$ together with SPE. Both SPE and UML-$\Psi$ are methods used within the industry for performance evaluation. SPE is mature, while UML-$\Psi$ is a relatively new approach.

Both SPE and UML-$\Psi$ use different models for performance evaluation. In this thesis, a translation mechanism is suggested for using SPE models with UML-$\Psi$. This mechanism covers most of the translation. However, it couldn’t manage to provide the exact translation of expanded nodes and case nodes. But this is purely because of the use of a specific UML modeling tools and UML-diagrams (use cases, activity and deployment diagrams). Expanded nodes are used by SPE for a multi-step process. But With UML-$\Psi$, activity diagrams need every basic step to be mentioned. As far as the case node is concerned, for each use case a separate activity diagram needs to be drawn with specific tag values. It makes it hard to use case nodes. UML modeling tool support for UML-$\Psi$ doesn’t allow the drawing of case nodes.

A simple ATM example was implemented both using an SPE tool and UML-$\Psi$. When the results were matched, it showed consistency. So, it can be concluded that UML-$\Psi$ can be used with SPE. But at the same time, usage of UML-$\Psi$ is extremely difficult. One faces a lot of challenges when it comes to installation and usage of UML-$\Psi$. This is due to the reliance on old versions of UML modeling tools (i.e. ArgoUML V.12 and Poseidon V1.4). And this issue is suggested to be resolved in order to compete in the industry.

6.1 Future Work

In order to make UML-$\Psi$ mature and competitive to other performance evaluation tools, there is a need to address these issues.

Currently UML-$\Psi$ does not handle all type of UML diagrams. And it is quite a possibility that different users may be more comfortable with different kind of diagrams for expressing the same things. For example one can find sequence diagram more interesting and easy to use as compared to the activity diagrams. Thus, the UML-$\Psi$ could be extended to include more types of UML diagrams. In particular, it would be very useful to include sequence diagrams and state charts in the performance modeling approach.
State chart diagrams are defined in the UML standard as a superset of activity diagrams [32]. However, state charts put more emphasis on the notion of “event”, while activity diagrams put more emphasis on the notion of “activity” taking some amount of time to complete before starting the next one. The simulation model should be easy to expand to include these different views of the system [2, p.134].

Other possible improvements may be related to the performance indices which are currently computed by the UML-Ψ tool. At the moment, only a limited set of measures is computed (i.e. Utilization of resources, Throughput of resources and Mean execution times of actions and use cases); more performance indices (for example distributions) would certainly be useful. The UML performance profile already defines some of them, so it is just a matter of extending the simulator. The libcppsim library can also be extended both by providing a richer set of statistical data analysis functions, and by providing some more high level simulation entities on top of the simulation process abstraction in order to facilitate the modeling process.

This work verifies the hardware resources in particular. A further possibility could be of extending this work for software resources (as used in SPE) and combination of both software and hardware resources.

Another definite improvement could be of making UML-Ψ compatible with at least the latest versions of Poseidon [19] and ArgoUML [20]. The new versions have added some new base classes for stereotypes, which are needed to be handled in UML-Ψ. Currently, it selects a base class (i.e. ModelElement), which changes it from the original base class and this leads to changing the results too (i.e. every base class has different tag values.).
References


