SMEPP: a novel System Modeling Environment for Performance Prediction

Master thesis by Maarten Vijfhuize (mvijfhuize@voogd.com)

Leiden Institute of Advanced Computer Science: dr. E.M. Bakker (erwin@liacs.nl)
Voogd & Voogd Diensten BV: ir. J.F. Spiegels (sspiegels@voogd.com)

Abstract

SMEPP is a novel System Modeling Environment for Performance Prediction. The environment consists of methods for modeling the entire context of a system, from hardware and software components to the workload posed on them. Additionally, it includes an application to simulate these models. SMEPP enables one to quickly and accurately assess the performance of a (proposed) system based on a minimal amount of input information. Also, the consequences of updates within the hardware and software components and the evolution of workload can be evaluated. Examples include increasing resource capacity, decreasing a components execution time, increasing the number of requests and so on. Particularly because of its wide context, SMEPP is applicable in real life situations. This will be illustrated by various cases. The first part of the experiments is aimed at the validation of SMEPP with respect to concrete scenarios. The remaining experiments are focused on prediction of the results in more abstract situations. It will be shown that SMEPP is a valid, scalable and general approach to system modeling and performance prediction.

Acknowledgements

This article was written in the period from September 2008 to May 2009. First of all, I would like to thank dr. Erwin Bakker for his indispensable support, advice and time during this period. Thanks also go to the people at Voogd & Voogd Diensten, for providing me with the chance to apply and verify the results of my research in significant real life cases. Special thanks go to everyone there who has contributed in any given way. In particular I would like to thank Sjaak Spiegels for allowing me to choose my own path in this project and to fully concentrate on this work. Also, our numerous discussions were a great source of inspiration.

Contents

1 Introduction

The term Software Performance Engineering (SPE) is somewhat misleading in that it seems to describe the process of performance analysis of software exclusively. SPE however represents a much broader field of research, as software performance is affected by many other system aspects, for example workloads and hardware capacity. Instead, the name refers to the ideal situation in which performance engineering is completely integrated within the software engineering process, as described by Smith and Williams [1]. Their approach is model-based, which means that performance models are created in the early stages of software development and the results are used to improve the architecture in the process. There is also a measurement-based approach of SPE, for example described by Barber [2;16], that addresses performance late in the development process, when the system is already functional. Ideally, one should try to combine the 'best of both worlds' in such a way that the measurement results assist in improving the models. However, the reader should keep in mind that both SPE approaches, as described in literature, are highly theoretical. In reality, true integration of performance engineering within software engineering is seldom. The reason for this is the gap between functional requirements (the usual concern of a software engineer) and non-functional requirements, such as performance, reliability and security [13]. Often, the development process focuses on functional correctness only. When that goal is accomplished, the application is fine-tuned for performance reasons. This is referred to as the 'fix it later' approach. However, the impact of these late modifications is small compared to architectural changes that might be applied when performance engineering is done earlier in the development process [1]. In some cases, non-functional properties are not addressed at all and performance evaluation is done when the system is already running in a production environment. This limits the possibility to make structural changes in order to optimize the performance when the results are unsatisfactory.

As said before, performance is not limited to the application itself. Instead, it addresses the complete environment in which software is running, from hardware resources to user behaviour. Thus, to create an appropriate performance model, one has to find a way of modeling all system aspects. Modeling methods, such as Use Case Maps (UCM) [3] and even the nowadays widely adopted Unified Modeling Language (UML) [4] are software oriented and lack features to effectively describe non-functional requirements in a wider context. In the case of UML, various profiles have been developed to improve the usability of the language in an SPE process. These include, in order of appearance, the UML Profile for Schedulability, Performance and Time (SPT) [5], the UML Profile for Modeling QoS and Fault Tolerance Characteristics and Mechanisms [6] and the planned UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE) [7]. See [9] for a comparison of the SPT and QoS profiles. From the systems engineering perspective, another interesting development is the Systems Modeling Language (SysML) [8], a subset of UML 2 with extensions for systems engineering that can be used to model hardware, software, information, personnel, procedures and facilities. Examples of its application are described in [31;32;33]. Note that all UML extensions mentioned here are specified by the Object Management Group (OMG), the organization also responsible for the official UML specification. From the architectural point of view, the

ArchiMate [19] language (also UML based) is worth mentioning. It started in 2002 as a research project by a joint effort of several Dutch companies (Ordina, ABN AMRO), institutes (CWI, TI) and universities (Leiden, Nijmegen) and was adopted by the standardization organization The Open Group in 2008. ArchiMate allows for integrated modeling of the architecture on three different levels (business, application and technology) which supports the decision making process across organizations. It was successfully applied in various cases [19;29;30] at Dutch organizations, for example UWV, Belastingdienst and SVB. Languages as SysML and ArchiMate are certainly promising. However they are not as widely used as their parent language UML. This means that there are not as many successful implementations known and that support and confidence for such an approach may be too low. Earlier attempts, for example the UML SPT profile, have been the basis for many projects [36;37] by now, but are significantly less powerful than for example SysML. This dilemma is probably also preventing a definitive breakthrough into a certain direction. As a result, new standards keep being proposed and the original goal of a powerful, universal modeling language for complete systems is farther away than ever [13].

Design models, both UML (possibly with an extension mentioned above) and non-UML, are usually translated into performance models to enable quantification of performance issues. The results are then used to improve the original design. A wide range of performance model formats exist, including stochastic Petri nets (Merseguer [11], King and Pooley [10]), stochastic process algebras [46], layered queuing models [1] and simulation models [47]. An even wider range of translations from different design models into different performance models exists. This issue was tackled by Woodside, Merseguer and others with the PUMA (Performance by Unified Model Analysis) [12] architecture. As part of PUMA, an intermediate format called Core Scenario Model (CSM) is proposed that can be used to easily translate various design models into various performance models within a single architecture. For capturing performance information, CSM depends on the UML SPT profile, which is originally developed for UML 1.4. Another drawback is that CSM is not the only intermediate format known. For example SysML depends on the XML Metadata Interchange (XMI) [15] standard for exchanging information between different environments, others use the Performance Model Interchange Format (PMIF) [14] and so on. Unfortunately these issues have prevented either of them from becoming the standard in software performance modeling.

The measurement-based SPE approach [2;16] is a vital method to either confirm or correct the results of the model-based methods. However, this approach also has its shortcomings. On of the main drawbacks is that in literature, often a significant amount of input data, for example server log files that describe a representative workload, is assumed to be available. In practice, this assumption is far from realistic, because in most cases this information is either missing or incomplete [16]. Several excellent techniques for performance prediction and capacity planning exist [16;17]. However the absence of accurate input data makes the results of even the best methods highly questionable. Even if test results are reliable, there is a lack of theoretical justification for improvements based on them [18]. For example, if a certain bottleneck is solved by increasing local capacity, it could easily introduce a bottleneck somewhere else in the system. One still

depends on an expert to analyze the results in each specific case before any changes can be implemented. This process is time consuming and hard to automate.

Summarizing, one can identify these main problems in the system performance field:

1. Modeling languages and performance evaluation methods suffer from a lack of standardization which prevents their successful use on a larger scale.
2. These languages and methods are often restricted to a certain context (software, hardware), which is too small to model and evaluate complete systems. This prevents their application (and justification) in larger, real life cases.
3. Usually there is too little accurate input information to effectively evaluate a system before it is deployed in a production environment.
4. There is a historical gap between evaluation of functional (does it work?) and non-functional (does it perform?) requirements which prevents their successful integration.
5. There is a lack of knowledge about the general consequences of modifications within hardware and software systems.

SMEPP is focused on the first three problems. Existing techniques (1) are applied to create a modeling environment for the entire context of a system (2), based on a minimal amount of input information (3). The goal is to enable quick and accurate assessment of system performance. Additionally, the approach should be applicable for a large part of the currently existing systems. These include web-based applications, e-commerce, multimedia services, enterprise management systems and so on [45].

Section 2 provides background information about some of the techniques that are applied. The general approach of SMEPP is described in Section 3. In Sections 4 to 7 an analysis of the components within the system context is given. The most important properties and relations within each component are determined, as well as how to model them. In Section 8 an application to simulate these models is proposed. Section 9 describes the experimental setup used to validate SMEPP with respect to real life and to determine its prediction accuracy. Using this setup, extensive experiments are done, aimed at both validation and prediction. The results are described in Section 10. Finally, Sections 11 and 12 contain conclusions and future work, respectively.

2 Preliminaries

2.1 UML diagrams

A UML d*eployment diagram* [4] gives a static view of the runtime configuration of a system. In other words: it shows the hardware within a system, the software running on that hardware and the connections between these components. Physical (hardware) components in a deployment diagram are labeled with the stereotype *Device*, while non-physical (software) components are labeled with *Artifact*. The software components use the same notation as in a UML c*omponent diagram*.

2.2 Continuous probability distributions

An *exponential distribution* is often used to model the time until failure of a device, or the time between consecutive events in a memoryless process (i.e. the time elapsed already has absolutely no meaning with respect to the remaining time until the next event). It is related to the *Rayleigh distribution* in that both are special cases of the more general *Weibull distribution*, whose reverse is also referred to as *type III extreme value distribution*. Another type of generalized extreme value distribution (more specifically, *type I*) is the *Gumbel distribution*. The *normal* or *Gaussian distribution* is used to model variables with a symmetric structure. In practice many natural phenomena can, to a certain degree, be approximated by a normal distribution due to the *central limit theorem* [42]. A related distribution is the *log-normal distribution*, where not the variable itself, but its natural logarithm is normally distributed. The *logistic distribution* resembles the normal distribution but has heavier tails. A *uniform distribution* is the best model if only a variables extremes are known and all values in between are more or less equally likely. If a distinctive mode (most likely value) is discovered, it is better to use a *triangular distribution*. The *u-quadratic distribution* is a useful model for symmetric data with two different modes. However, the *Beta distribution* allows more flexibility with respect to non-symmetric data and non-quadratic shapes of the probability density function. The *Gamma distribution* is frequently used as a model for waiting times. The *Chi-square distribution* is a special case of the Gamma distribution. Further details for these distributions are given in [41][43][44].

In addition to the probability density function (PDF), all distributions mentioned above have a *cumulative distribution function* (CDF) which describes the probability of a value lower than or equal to x occurring, according to the distribution concerned. An overview of these cumulative distribution functions is given in Appendix B.

2.3 Goodness of fit

A *goodness of fit test* assesses how well a given data set fits a certain probability distribution. Many goodness of fit tests are known, some general and some specific for certain distributions. The *Kolmogorov-Smirnov test* [44] finds the maximum distance between the empirical CDF of a data set and the theoretical CDF *F* of the distribution

being tested. That is, for each value Y_i in the ordered set, it compares the probability of a value lower than or equal to Y_i in the set with its theoretical probability according to F and determines the maximum difference between them:

$$D = \max_{1 \le i \le n} \left(F(Y_i) - \frac{i-1}{n}, \frac{i}{n} - F(Y_i) \right)$$

If the data set corresponds to the distribution being tested, D will converge to 0 almost surely. The *Anderson-Darling test* [38] assesses whether there is evidence that the data set does not correspond to the distribution concerned by measuring the distance between them as:

$$A^2 = -n - \frac{1}{n} \sum_{i=1}^{n} (2i-1)(\ln F(Y_i) + \ln(1 - F(Y_{n+1-i})))$$

where n is the number of values in the data set, Y_i for $i = 1 \ldots n$ are the values sorted from low to high and F is the CDF of the distribution being tested. The value of A^2 should be slightly adjusted for low n values and compared to the critical value for the distribution being tested. The specifics of the test for different distributions have been published in different papers by Stephens, e.g. [39].

A correlation coefficient is a measure of correlation between two variables. One speaks of correlation when a certain linear relationship exists between the variables. The Pearson product-moment correlation coefficient [40] is the most widely used correlation coefficient. It is equivalent to the covariance of variables X and Y divided by the product of their standard deviations:

$$\rho(X,Y) = \frac{\operatorname{cov}(X,Y)}{\sigma(X)\sigma(Y)}$$

The value of ρ may vary between -1 and 1. A value of -1 or 1 means that there is a perfect (inverse) linear relationship between X and Y. A value of 0 means that there is no linear relationship at all. There may however be another type of relation.

3 Solution approach

3.1 Introduction

With respect to the standardization problem mentioned in Section 1, it seems obvious to select the most widely adopted methods rather than creating new ones, because that would only make the situation worse. For a modeling language the initial choice is UML, while as an intermediate format the XMI standard seems to be sufficient. The resulting formalism should allow one to model all system aspects that might be involved in its performance, in order to bridge the gap between the software and hardware perspectives. SMEPP will be used to predict the consequences of changes in any of these aspects for the system as a whole, based on knowledge gathered and refined in the process. For example, if a certain application request can be handled twice as fast, what is the result for the complete system? Or, if the number of requests is doubled, what happens? SMEPP may also be used to address the performance within a different environment, which is useful for example when predicting the result in a production environment based on measurements in a test environment. Finally, all of this should be possible even with a minimal amount of information available, to allow for quick experiments.

3.2 Model components

Within the system context described, the following components are identified:

- workload (Section 4)
- hardware (Section 5)
- software (Section 6)

SMEPP will quantify the system performance level based on these three components. Therefore each of them needs to be modeled, as shown in Figure 1. The performance level might for example be defined as the distribution of execution times. Given a (satisfactory) target performance level, SMEPP may also be used to answer capacity and scalability related questions by determining for example:

- the minimum hardware required;
- the maximum workload allowed;
- the minimum software efficiency needed.

For this purpose, the environment should include simulation possibilities. Several simulators for UML models exist [26;27;28;34;35], however they are usually applied within a restricted context. For example, they simulate a UML description of an application by running physical processes on a system. Within SMEPP both the application and the system (and finally the workload) can be simulated, thereby making the result independent of the system the simulation is run on. See Figure 1. This adds some layers of abstraction, which is why a novel simulation engine is introduced in this article. Naturally, there should be a trade-off between model simplicity and simulation

accuracy. Software characteristics for example, can initially be abstracted by considering an application as a 'black box' with only basic properties. This representation can be made more concrete if needed. Additionally, an initial model might be limited to a single system running a single application, after which it is iteratively extended to cover more of the environment.

Figure 1 shows a schematic overview of the SMEPP approach. The red square represents the focus of this article: modeling methods and a simulation engine.

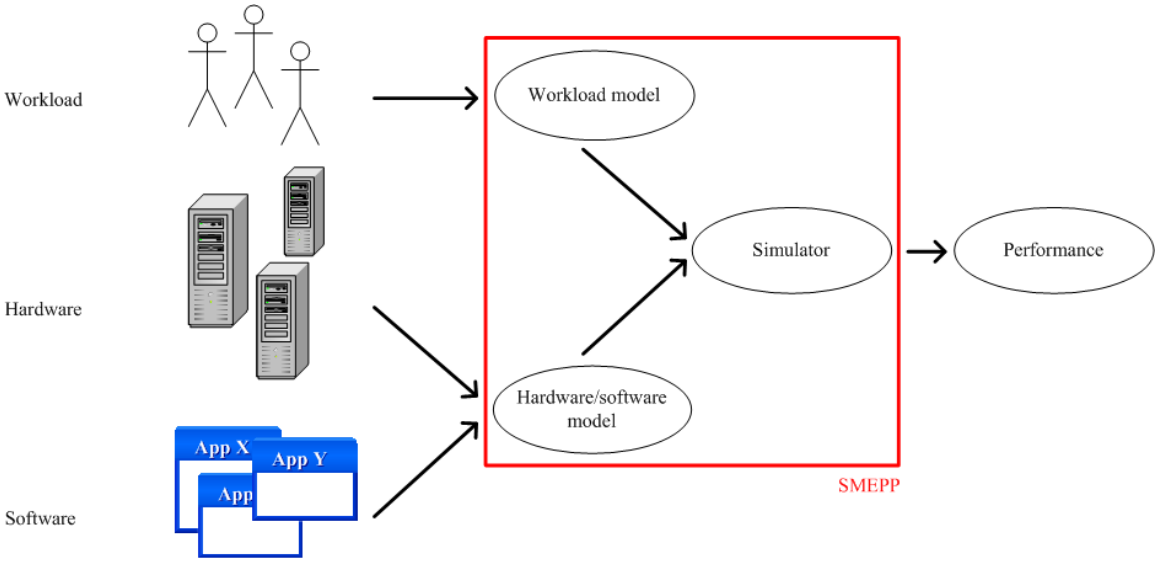

Figure 1: SMEPP context and approach

4 Workload analysis

4.1 Introduction

In this section, a workload will be defined and characterized by its most important properties. The findings will result in a representation for the workload component within SMEPP. An example analysis is given in Section 10.3.

4.2 Definition

In this article, workload is referred to as the amount, type and distribution of requests posed on a system within a certain period of time. Thus, workload does not say anything about the system itself or its performance. A workload is typically captured by a log file, on either system or application level, that describes what, when and how many requests are posed on the system or application concerned.

One way to describe a workload is to describe the *request rate* over a period of time. The period that is best used to describe such a workload depends on the situation. Certain workloads may be described best over a period of 24 hours (for example, web site traffic) while for others such a workload description is not representative (for example, a web site that has more visitors during weekdays than in the weekend).

Another way to define a workload is to focus on the *request interval*, the time between the start of consecutive calls to a system or application. The request interval becomes more meaningful as the resolution increases. For example, an average request interval of 10 seconds over 24 hours is useless if the average was 5 seconds during the day and 15 seconds during the night. However, if the average request interval would be measured every hour, the result would be far more meaningful (and accurate). Remember that increasing the resolution actually means decreasing the measurement interval.

While both the request rate and the request interval seem to be sensible metrics for describing a workload, there are some pitfalls. When a request rate is visualized in a graph, it seems logical to scale the values linearly along the axis. In fact the metric should be interpreted non-linearly. For example, the impact of increasing the rate from 1 to 2 requests per minute is far greater than the impact of increasing the rate from 20 to 21 requests per minute. The first scenario means decreasing the interval from 60 to 30 seconds while the second corresponds to decreasing the interval from 3 to 2.86 seconds. One has to keep in mind that both metrics are not linearly related (Figure 2).

Both request rates and request intervals have advantages and disadvantages. In general one could argue that the request rate is best used to visualize a workload. Consider a graph that relates the request rate to the time. The surface beneath the graph represents the total number of requests in the depicted period. Such a representation is easy to interpret. Contrary, if the request interval is considered in relation to the time, the surface meaning is less obvious. However, the interval more realistically represents the impact of

a changing workload for the system concerned (recall the example from Figure 2). This makes the request interval (or its distribution) more valuable as input for analysis and simulation. Thus, the request rate will be used for visualization purposes, while the request interval will be used as input for the experiments later on.

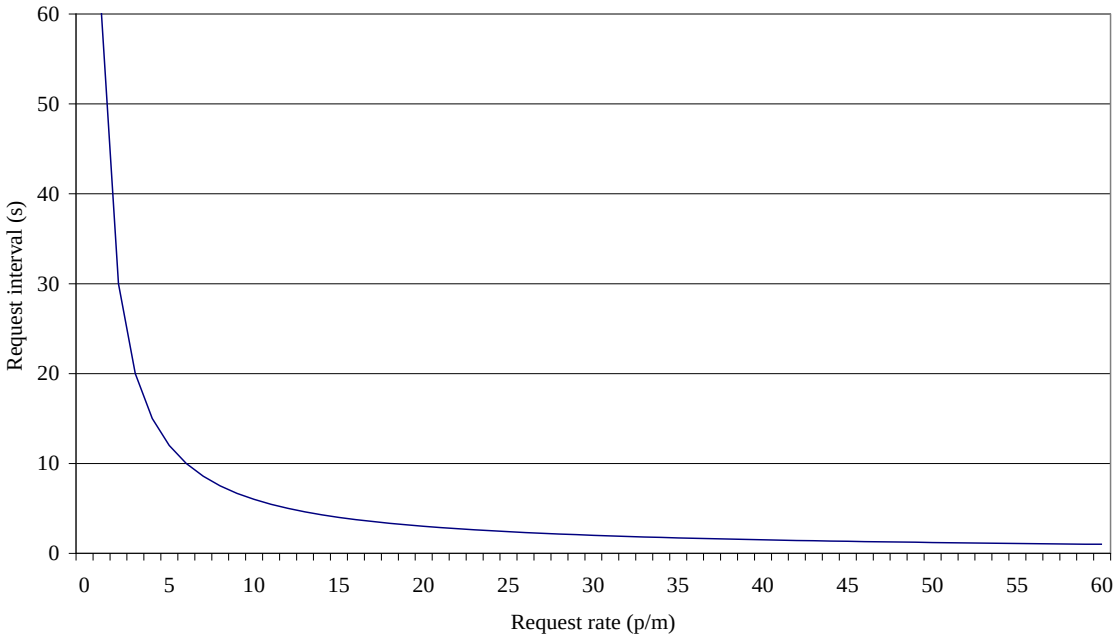

Figure 2: relation between request rate (per minute) and request interval (in seconds)

For both request rate and request interval there are 3 choices to be made. Optimal settings for each of them strongly depend on the specifics of the situation.

- Source window (SW): the period over which the source data runs
- Target window (TW): the period used to describe a workload
- Resolution (R): the number of measurements within the target window

4.2.1 Source window

Increasing the source window will improve the chances that extremes are included in the representation. However, a too large source window may result in an unrealistic workload with respect to the situation that needs to be modeled.

4.2.2 Target window

Decreasing the target window is useful for visualization, for example to show a part of the window in more detail. However, it does not improve the accuracy of the workload representation. Increasing the target window may improve the accuracy, because a small target window might not show the evolution of values within a larger window. Example: a target window of one day when all weekdays are different. When the target window is smaller than the source window, there may be multiple values for each data point in the

target window. In those cases, aggregate functions such as minimum, maximum and average should be used to represent the values.

4.2.3 Resolution

Increasing the resolution (to a certain extent) will result in a more realistic view and more correct interpretation of extremes. In general one could argue that it is worth increasing the resolution if the results at the current resolution are generally larger than the number of smaller time units within the current resolution. Remember that for a request rate, it does not make sense to use a resolution other than the one used to define the rate. Example: it does not make sense to measure the number of requests per hour every minute, or just once a day. Hence, a trade-off should be made between resolution and accuracy.

4.3 Distribution of requests

Until now, a workload was considered as a concrete log file. To model a workload, a more abstract notation is needed. This can be achieved by determining probability distributions that approach the values in a log file. As said before, the focus will be on request *intervals* for simulation input, which means that *continuous* probability distributions are required. Examples and their occurrences are given in Section 2.2. Recall, in the case of modeling request rates, *discrete* probability distributions would be needed.

4.3.1 Dynamic optimal resolution

Before determining the probability distributions that approach the values in the log file, a correct resolution has to be selected. Remember that when using a low resolution, the resulting model is simple, however the specific distributions of certain parts within the window may miss. Using a high resolution leads to a very accurate representation, however goes beyond the idea of a model. Thus, the optimal resolution is a trade-off between accuracy of the representation and simplicity of the model. In addition to that, the optimal resolution may be different throughout the window. Hence, a method of determining a *dynamic optimal resolution* is needed. This can be done by using a binary algorithm that takes a set of intervals, splits the set in 2 subsets, determines if the characteristics of the subsets (for example: mean and variance) are substantially different from the original set and if so, recursively continues with the subsets. Using variable lower bounds on the set size and the deviation of subsets from original sets, enables one to make the right trade-off mentioned earlier. See the next page for a pseudo code example.

4.3.2 Best matching probability distribution

Once the characteristic parts of the scenario have been determined, each part can be represented with the probability distribution that best matches the values within that part. This can be done by using statistical methods like the Kolmogorov-Smirnov test (K-S)

and the Anderson-Darling test (A-D). Both are described in Section 2.3. The K-S test is typically used to determine if a given data set corresponds to a certain distribution by measuring the maximum distance between the empirical and theoretical CDF and comparing this to a certain critical value. Hence, this is a Boolean test. However, the data should be compared to a set of different distributions in order to select the best approximation. In that case, the maximum distance is less indicative of the correspondence and it would be better to sum the distances, like the A-D test does. However, the A-D test is only valid for a few specific distributions, while the K-S test may be used for any continuous probability distribution for which the CDF is defined. To overcome these issues, the best of both worlds is combined. In fact, for each value the distance between the empirical and theoretical CDF is measured (like the K-S test) and these distances are summed (like the A-D test):

$$D' = \sum_{i=1}^{n} \left| \frac{i}{n} - F(Y_i) \right|$$

D' is determined for a set of different distributions (an overview is given in Appendix B) and the distribution for which *D*' is minimal, is selected as the best approximation. This procedure is repeated for each characteristic part in the log file, according to the dynamic optimal resolution algorithm:

```
dynres (INPUT, MIN_SIZE, MAX_DEVIATION)
{
    OUTPUT = empty;

    if length(INPUT) >= 2 * MIN_SIZE
    {
        LEFT = first half of INPUT;
        RIGHT = second half of INPUT;

        if absolute(mean(LEFT) - mean(INPUT)) / mean(INPUT) > MAX_DEVIATION
        {
            add to OUTPUT: dynres(LEFT, MIN_SIZE, MAX_DEVIATION);
        }
        else
        {
            add to OUTPUT: LEFT;
        }

        if absolute(mean(RIGHT) - mean(INPUT)) / mean(INPUT) > MAX_DEVIATION
        {
            add to OUTPUT: dynres(RIGHT, MIN_SIZE, MAX_DEVIATION);
        }
        else
        {
            add to OUTPUT: RIGHT;
        }
    }
    else
    {
        add to OUTPUT: INPUT;
    }

    return OUTPUT;
}
```

Pseudo code for dynamic optimal resolution algorithm

4.4 Workload model

The procedure described in the previous section results in a list of parameterized distributions and their lengths, which looks like this:

```
length;distribution(parameter1,parameter2)
length;distribution(parameter1,parameter2,parameter3)
length;distribution(parameter1)
...
```

where length $\in \mathbb{N}$, distribution $\in$ {Normal, LogNormal, Logistic, Beta, Gamma, ChiSquare, Gumbel, Exponential, Rayleigh, Uniform, Triangular, UQuadratic, Constant}.

Using a workload model instead of a concrete log file has several advantages. First, different simulations of a single model will result in comparable, however different scenarios. This may help to discover (worst case) situations one would not have seen if a log file (i.e. fixed scenario) was used. Second, a workload model can be easily modified, in order to simulate the consequences of an evolving workload. More specifically, the parameters of the probability distributions allow for easily increasing or decreasing (parts of) the workload based on real life observations. It is a lot harder to realistically increase or decrease a workload based on a fixed scenario. Finally, a workload model may easily be specified manually as well, without being based on real life data. This allows for fast and dynamic experimenting.

4.5 Correctness

The described model includes the request interval distributions and parameter values. These are considered as input. Based on that, SMEPP will simulate the performance, in terms of execution times. These are considered as output. This suggests that the intervals partly determine the execution times. However there may also be a backwards relation, in the sense that execution times (output) may partly determine the intervals (input) that will follow. For example, a high execution time might prevent a user from issuing a new request for a while, or from issuing a new request at all. Hence, it has an indirect influence on the following intervals. The question is how significant this influence is.

When the interval distributions are based on real life data (i.e. on a log file), a possible influence of execution times on intervals is already taken into account. However, when the distribution parameters are modified in order to increase or decrease (part of) the workload, this factor might come into play. Consider the case in which the interval distribution parameters are modified such that the mean interval will be twice as low (in the period concerned) and as a result the execution times will increase. Indirectly, the intervals of an individual user may *in*crease, due to waiting time, while the goal was to *de*crease them.

This phenomenon is illustrated in Figure 3. The horizontal line represents time, while the vertical lines each represent a point in time at which a request is issued. Hence, i_x

represents the interval between requests x and x+1, e_x represents the execution time of request x and u_x represents the user issuing request x.

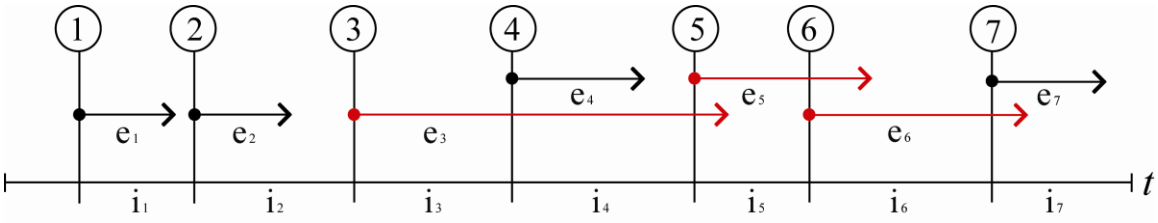

Figure 3: relation between request interval and execution time

The assumption is that e_x may depend on multiple factors:

- e_x depends on i_x if $e_x > i_x$
- e_x also depends on i_{x+1} if $e_x > i_x + i_{x+1}$
- and so on, in general:
- e_x depends on i_{x+a} if $e_x > \sum_{y=0}^{a} i_{x+y}$ with $a \geq 0$

However to what extent the opposite is true? In other words: does i_{x+a} also depend on e_x? When e_x is greater than i_x (illustrated by the red arrows) i_x is obviously not caused by e_x. However one of the following i_{x+a} ($a > 0$) may depend on e_x provided that request x+a+1 is issued by the same user. Note that while an execution time may depend on multiple intervals, an interval may only depend on a single execution time. Hence:

- i_x depends on e_x if $e_x < i_x$ and $u_x = u_{x+1}$
- i_{x+1} depends on e_x if $i_x < e_x < i_x + i_{x+1}$ and $u_x = u_{x+2}$
- and so on, in general:
- i_{x+a} depends on e_x if $\sum_{y=0}^{a-1} i_{x+y} < e_x < \sum_{y=0}^{a} i_{x+y}$ and $u_x = u_{x+a+1}$ with $a \geq 0$

Naturally, e_x also depends on any overlapping e_y with $x \neq$ y. Given that $u_3 = u_6$, $u_4 = u_5$ and $u_1 \neq u_2 \neq u_3 \neq u_4 \neq u_7$ the dependencies in Figure 3 are as follows:

- e_3 depends on i_3, i_4, e_4 and e_5
- e_4 depends on e_3
- e_5 depends on i_5, e_3 and e_6
- e_6 depends on i_6, e_5 and e_7
- e_7 depends on e_6
- i_5 depends on e_3
- i_4 depends on e_4

The i_x values corresponding to the red e_x arrows in Figure 3 are perfectly valid, as long as during execution of request x, subsequent requests are issued by different users or the system is asynchronous (or both). However, if this is not always the case, using a custom

interval distribution (i.e. not based on real life data) may lead to unrealistic scenarios. For example, if Figure 3 corresponds to a synchronous system (i.e. the user has to wait for a request to finish before issuing a new one) and requests 3 and 4 are issued by the same user, this scenario is invalid. This needs to be taken into account upon simulation of the model. More specifically, when request x is started, the probability that within the next interval i_x any request y will be finished with $y \leq x$ and $u_y = u_{x+1}$ needs to be determined. Hence, the probability depends on the number of unique users and the execution times seen so far. For a synchronous system the generated intervals need to be corrected according to this probability. In general, increasing execution times or decreasing the number of unique users within a scenario will result in an increased probability of intervals depending on execution times.

5 Hardware analysis

5.1 Introduction

To model a hardware component (more specifically: its performance) one needs a few metrics that together define its relative speed. An ideal method to gather these metrics is to run a fixed set of benchmarks that generates exactly the same amount of work on each component. In this section, such a set of benchmarks will be proposed. For a computer system, the benchmarks should assess different resources as isolated from each other as possible. Only then it is possible to predict the consequences of a modification in a single resource for the system as a whole. Also, the benchmarks should exploit the full power of resources to determine their theoretical performance limit. Finally, the set of benchmarks should be as representative as possible with respect to the software components of the model (Section 6). An example analysis is given in Section 10.4.

5.2 Benchmarks

The Stream memory benchmark [20] is designed to work with data sets larger than the available cache on any given system, so that the results are indicative of the memory performance only and that the influence from other resources is as low as possible. It reports memory bandwidth in megabytes per second. The Dhrystone CPU benchmark [21] is commonly used to test integer arithmetic. The algorithm is based on the C implementation written by Reinhold P. Weicker. The test reports in MIPS (Million Instructions Per Second). The Whetstone CPU benchmark [22] is commonly used to test floating point arithmetic. The algorithm is based on the C implementation written by Rich Painter. The benchmark is designed to test the speed of commonly used floating point CPU instructions. The test also reports in MIPS. The Compression benchmark uses an Adaptive encoding algorithm based on source code from Witten, Neal and Cleary [23]. The system uses a model which maintains the probability of each symbol being the next encoded. It reports a compression rate of 363% for English text, which is slightly better than the standard Huffman method. This test reports its results in compressed kilobytes per second. The Quicksort benchmark [24] continually sorts 1000 strings of 256 characters. The test reports the speed of the sorting in thousands of strings per second. The Blowfish benchmark uses the equally named enciphering algorithm [25]. It is based on the C implementation written by Paul Kocher. Data is enciphered using a 16 byte key in blocks of 4 KB. The test reports in encrypted kilobytes per second.

5.3 Hardware model

Within SMEPP, a system is represented by the benchmark results of each resource that is used by one or more of the applications represented in the same model. These values should be defined in MIPS (Million Instructions Per Second). Depending on the resource, MIPS might refer to CPU instructions, memory instructions and so on. Section 7 describes how to incorporate these values in a UML model.

6 Software analysis

6.1 Introduction

This section will describe a method of modeling software components (more specifically: their utilization of resources) in the same terms of the benchmarks that were selected in Section 5. An example analysis is given in Section 10.4.

6.2 Optimal utilization/time

The first step is to look at the minimal resource utilizations for different phases in an applications process. This is done because these values are most representative for optimal application performance with minimal influence from other applications or environment factors. When these tests are done on a baseline system that is completely utilized by the application concerned, the number of instructions needed for the application to finish its job may be estimated, by multiplying the theoretical benchmark speed of each resource with the optimal execution time measured. When benchmark values cannot be determined (for example for external systems) the focus will simply be on optimal execution times.

6.3 Individual efficiency

Next, the same experiment should be run on several other systems, which may or may not be fully utilized by the application. In the process, their optimal execution times should be recorded and related to the base line execution time. Also, the benchmarks of those systems should be related to the base line benchmark. By dividing the execution time speedup with the benchmark speedup, efficiency values can be determined for all systems. They represent the part of the theoretical speed that was actually used by the application. Based on these values, an individual efficiency trend E_i can be discovered:

$$E_i(x,c) = \frac{\left(\frac{t_c}{t_x}\right)}{\left(\frac{b_x}{b_c}\right)}$$

where x is the system under test, c is the base line system, t_i is the optimal execution time for system i and b_i is its benchmark value. For an ideal application, E_i should always evaluate to 1, meaning that the theoretical speed is fully utilized and that application speed increases linearly with it. The individual efficiency expression, whether it is a simple constant or a complex function, is an important property for representing applications within SMEPP. In fact, it does not really matter what benchmarks are used to represent the systems, as long as an E_i expression can be defined, that correctly relates the benchmark values to the execution times on those systems. In practice however, simple E_i expressions and thus, representative benchmarks are preferred.

6.4 Multi tasking efficiency

The simplest way to model multi tasking behavior is to divide the number of instructions that a resource (for example CPU) could execute within a certain period of time by the number of operations executed in parallel by the same resource. This way, all operations receive an equal 'slice' of execution time within each execution cycle. Remember that this doesn't mean that each operation processes the same number of instructions within a cycle, because operations may have different efficiencies (as described in Section 6.3), meaning that they will only process part of the instructions they could process in theory. By default operations may run in parallel according to this model. However by limiting the number of parallel operations to one, a simple sequential resource can be modeled.

The time slicing model described here is a generalization of several more distinctive scheduling principles and thus, applicable in almost any case. However, its efficiency largely depends on the specifics of the resource and the characteristics of the applications running in parallel. For example, a dual core CPU will more efficiently run two tasks in parallel, provided that the application exploits these features. To incorporate these dependencies into the general time slicing model, an efficiency factor similar to the one described in Section 6.3 is introduced. However, this factor does not represent the efficiency of an individual task being executed with respect to the theoretical speed possible. Instead it represents the efficiency with which multiple tasks may run in parallel. More specifically, it represents the ratio of the optimal execution time of *n* consecutive equal operations and *n* simultaneous equal operations:

$$E_m(x,n) = \frac{nt_x(1)}{t_x(n)}$$

where $t_x(i)$ is the optimal execution time of *i* simultaneous operations on system *x*. Hence, if $E_m(x,n)$ is greater than 1, it means that *n* simultaneous operations are completed in less time than *n* individual operations on system *x* (i.e. that parallelism is exploited) and vice versa. Ideally this factor should approach the number of operations that can be processed in parallel without loss of performance. For example, for a dual core CPU, E_m should approach 2. In practice however there are lots of other factors, both with respect to the application and the system, that may determine why these ideal values are seldom reached.

6.5 Software model

Within SMEPP, an application is represented by its optimal utilizations, either in terms of instructions or time, with respect to different system resources (Section 6.2). Additionally, one may specify expressions that approach the individual efficiency E_i (Section 6.3) and multi tasking efficiency E_m (Section 6.4) for these resources. Here, curve fitting might be used. Both E_i and E_m default to 1. Finally, it is important to find the most important parameters of the application, determine the probability distributions of their values and find out how the previous observations (e.g. resource utilizations) relate to them. Section 7 describes how to incorporate all findings in a UML model.

7 Hardware/software model

7.1 Introduction

In this section a UML model will be described, to represent the hardware and software components within a system based on the findings from Sections 5 and 6. This model is based on an extended UML deployment diagram (Section 2.1). The main objects in the model are *Devices* (representing hardware components) and *Artifacts* (representing software components). Initially, only their names have to be specified. All other properties of devices and artifacts are defined using several types of *Constraints*. Finally, there are 3 types of *Relations* between objects.

7.2 Relations

A *Deployment* is a relation from an artifact to a device, meaning that the artifact is running on that device. Because artifacts may have multiple deployments, an artifact only has to be specified once, even if it runs on multiple devices. If multiple instances of the same artifact should run on a single device, multiple deployments can be specified between the artifact and device concerned

A *Dependency* is a relation from an artifact to another artifact, meaning that the former depends on the latter. In other words: the former cannot finish before the latter is finished. Artifacts may have multiple dependencies that may be deployed on the same or other devices. This way, requests to other devices can be incorporated. The dependent artifacts may have their own dependencies, which enables one to create a tree-like structure of artifacts.

A *Manifestation* is a relation from an artifact to another artifact, meaning that the former is an instance of the latter. Artifacts may have multiple manifestations. This is useful for example when the properties of an artifact (Section 7.3) should have different values in different cases.

7.3 Constraints

Constraints typically consist of a name and a value. An artifact *A* or device *D* may have several types of constraints associated with it:

- *Benchmark*: the theoretical number of instructions a resource within *D* may execute (Section 5.2). These values should be defined in MIPS. Benchmark names consist of a ‘b’ followed by the resource name.
  Example: bCPU = 308.8.

- *Utilization*: the minimal time (for example, in milliseconds) that *A* will utilize a resource within *D* (Section 6.2). The name refers to the resource concerned and

should be prefixed with ‘u’.
Example: uCPU = 988.

- *Time*: the range of time (for example, in milliseconds) that *A* needs to run. This is used when benchmarks and utilizations are unknown (Section 6.2).
  Example: t = 1746-2261.

- *Efficiency*: an expression that represents the individual efficiency E_i (Section 6.3) of an operation of *A* running on a resource within *D*, or the multi tasking efficiency E_m (Section 6.4) of operations of type *A* running on a resource within *D*. Both default to 1.
  Example: eCPU = $-0.0448\ln(x)+0.6656$

- *Capacity*: an upper bound on the number of requests that may run in parallel within *D* or the number of parallel executions of *A* (Section 6.4). Both default to infinity.
  Example: c = 1 (to represent a sequential resource).

- *Parameter*: the name of a parameter that *A* requires. It should be prefixed with ‘p’. A value is not necessary.
  Example: pProductCode.

- *ParameterValue*: a possible value for a parameter, associated with an instance of *A*. The name refers to the parameter, but should be prefixed with ‘v’ here.
  Example: vProductCode = 3.

- *Probability*: the chance that (an instance of) *A* will be executed at runtime. Defaults to uniform probability.
  Example: p = 0.11.

7.4 Example

Figure 4 shows an example of a hardware/software model, as it can be composed in any graphical UML environment. *Artifact1* and *Artifact2* are deployed on *Device1*, *Artifact3* is deployed on *Device2*. *Artifact1* depends on both *Artifact2* and *Artifact3*, in other words: a request to *Artifact1* cannot finish before those to *Artifact2* and *Artifact3* are finished. *Artifact3* has two manifestations. They are constrained with their probabilities and the parameter values associated. As a result, when *Artifact3* is called, the correct manifestation (*Artifact3a* or *Artifact3b*) is called according to the parameter value, which is determined by the probabilities set. The other types of constraints described above, are shown in the example too.

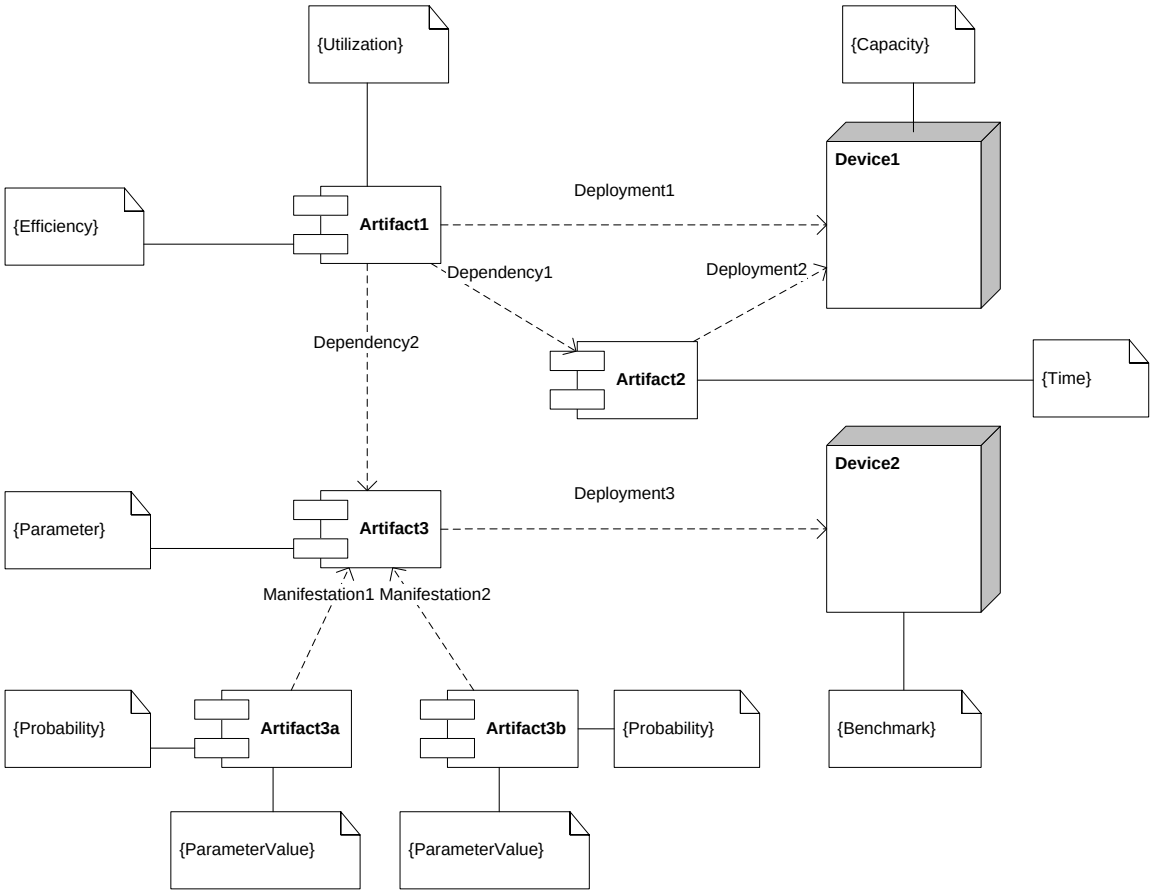

Figure 4: example UML hardware/software model

8 Simulator

8.1 Introduction

This section will introduce *XMISim*, an application used to simulate the UML hardware/software models proposed in the previous section and predict the performance of the represented system under various workloads. XMISim was written in the C# programming language. Part of the process involved the development of a library that can be used to generate random numbers corresponding to a wide variety of probability distributions (see Appendix B) and to estimate their parameters based on real life data. Appendix A contains a class diagram for the XMISim application.

8.2 Simulator input

A hardware/software model as described in the previous section can be composed within any graphical UML environment. However, to load the model into the simulator, it should be in XMI [15] format. Most UML environments provide an 'export to XMI' feature.

As described in Section 3.2, a third component is needed to simulate the model: the workload component. This component is defined separately, either by loading a log file describing a real life scenario, or by defining a workload model based on probability distributions. Log files should be in CSV (Comma Separated Values) format. Confusingly, the values are actually separated by semicolons. The first row contains the column headers, while each consecutive row represents an individual request. The CSV file may include an 'interval' column (specifying the time between a request and the previous one) and a 'deployment' column (specifying the artifact and device the request is associated with). Additional columns may be used to specify parameter values that should be used when simulating the request. The headers of these columns should correspond to parameter names as defined in the hardware/software model. The actual order of columns is irrelevant. None of the columns are required, because the distributions of intervals, deployments and parameter values may also be defined in the application itself. An example CSV file:

```
interval;deployment;parameter1
6436;Artifact1 @ Device1;5
7162;Artifact2 @ Device2;6
1749;Artifact1 @ Device1;6
1595;Artifact3 @ Device2;4
4550;Artifact1 @ Device1;5
7288;Artifact2 @ Device1;3
2752;Artifact3 @ Device2;6
...
```

When a CSV file is loaded, XMISim determines the distributions of the values in the file in order to generate a workload model. For the deployment and parameter columns, the probability distributions of their values are simply determined. For the interval column a more advanced method is used. The concept is explained in Section 4.3. The results of all

determinations are shown in the main window (see Figure 6) and can be modified by the user before starting the simulation. This way, the user has several options for simulating the intervals, the deployments as well as the parameter values. One may choose to

- exactly replay a log file;
- simulate a model of a log file in terms of probability distributions;
- simulate a modified or completely custom model;
- combine any of the above; for example: use the parameter distribution from a log file with custom intervals, and so on.

All this results in great flexibility with respect to workload specification and the possibility to compare different scenarios.

8.3 Running the simulator

When the XMI file is loaded, its structure of objects, relations and constraints is exactly recreated. In addition to that, some runtime components are added. For example, for every benchmark constraint associated with a device, a *Resource* is created that will process *Operations* at the rate specified. Each device also has a *Global* resource that will process operations of which the specific resource utilizations are unknown. These include the operations defined by time constraints. Figure 5 shows the example input model from Figure 4 in its runtime configuration.

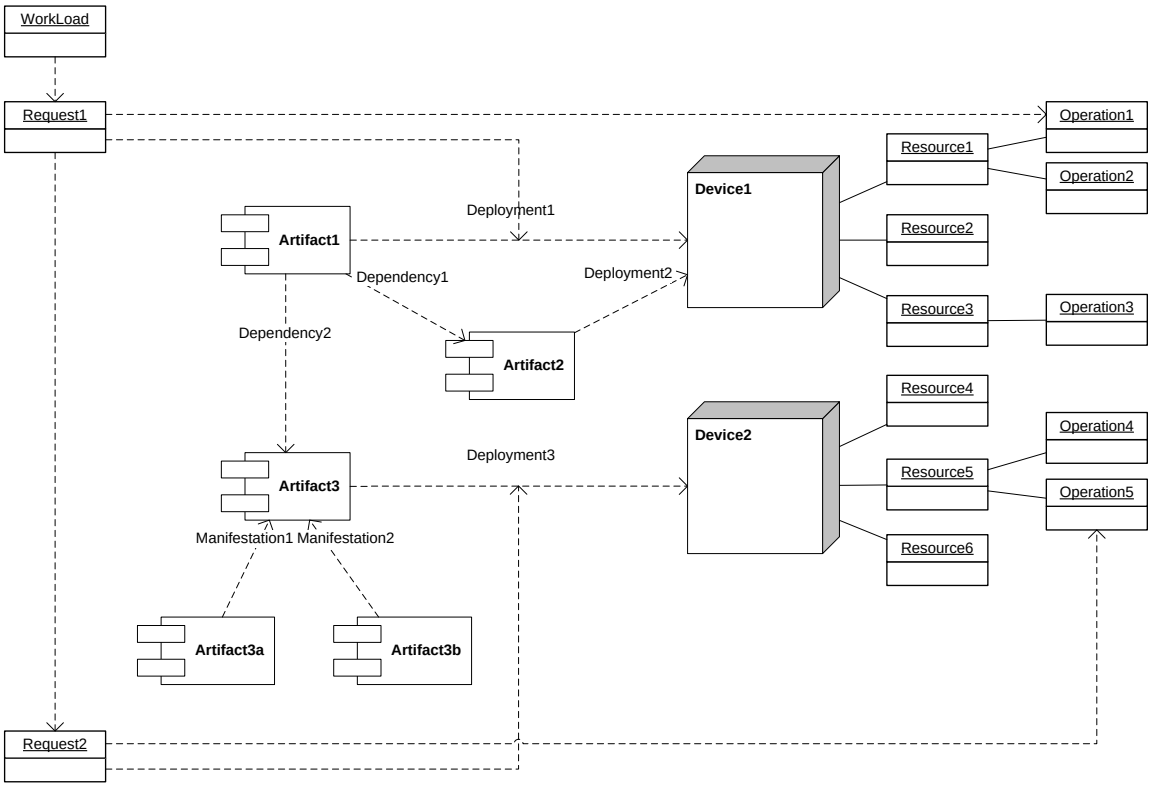

Figure 5: example model from Figure 4 at runtime

After starting the simulation, XMISim will create a number of *Requests* with intervals based on the workload description supplied. A request is associated with a deployment relation, thus with artifact *A* and device *D*. When a request is started, the resources of *D* are supplied with operations based on the utilization and time constraints of *A* (see Figure 5). When *A* requires a parameter, its value is either taken from the log file (if supplied) or selected from the possible manifestations by probability, or uniformly if no probability constraints are set. Next, the instance of *A* associated with the selected parameter value runs. This way, it is possible to simulate different utilizations and times for different parameter values. Based on the dependencies of *A*, requests for those artifacts are also created. A request is finished when all of its own operations and those of its dependencies are finished. More specifically, the time needed for an operation *O* to be completely executed by a resource *R* may depend on:

- the rate at which *R* processes instructions (benchmark constraint)
- the initial utilization or time needed for *O* with respect to *R* (utilization or time constraint)
- the part of the theoretical speed of *R* actually exploited by *O* (individual efficiency constraint)
- the influence of operations processed in parallel by R (multi tasking efficiency constraint)
- the maximum number of operations processed in parallel by *R* (capacity constraint)

Except for the benchmark, all of these constraints may have different values corresponding to different manifestations of an artifact. Additionally, the occurrence of these manifestations may be influenced by probability constraints. All of this leads to a dynamic simulation of a static hardware/software model.

8.4 Timing

XMISim has a built in timing mechanism with a default cycle time of 1 millisecond. However, the precision may easily be adapted if needed. Upon each cycle, the simulator will enable the resources to process some instructions, check if operations are finished and start new requests if needed according to the interval distribution. Generated intervals are corrected if they are unrealistic with respect to the number of unique users and the execution times seen so far. See Section 4.5 for details.

XMISim uses a simple time slicing principle as described in Section 6.4. Each resource has a list of active operations and a list of queued operations. The former contains operations that are being processed at that moment, the latter contains operations that, for example, have been started but cannot run because they have to wait for another operation to finish. Upon each cycle, the resource will check if the queued operations may already run and if so, move them to the active list. Next, the number of instructions the resource could theoretically process within that cycle, is divided by the number of active operations. All of them are allowed to process a ‘slice’ of the total number of instructions.

8.5 Efficiency

Remember that an individual efficiency constraint refers to the part of the theoretical speed that is actually used by an application. In real life, efficiency varies during execution of an operation. This is why peaks are typically seen when looking at, for example, CPU usage graphs. Although the efficiency of an operation being processed by a resource depends on both the operation and the resource within SMEPP, its value will be fixed throughout simulation of that operation. This means that there will be no peaks, unless multiple operations run in parallel. However, the surface beneath such a graph, corresponding to the number of instructions executed, is the same as in real life. This generalization makes it possible to simulate general resources without specific knowledge of the used CPU, memory and so on. The constraints described in Section 7.3 allow for more specific representation and simulation of different types of resources.

8.6 Output

Figure 6 shows a running simulation in XMISim. The left side is used to load XMI and CSV files and to define several parameters. In the middle column, the devices, resources and their current utilizations are shown. The right side contains the results of the files loaded and the running simulation. XMISim will report its results in the form of response time distributions of steps in the process as well as the total process. These values are easy to compare with real life observations (for validation purposes) and easy to interpret (for prediction purposes).

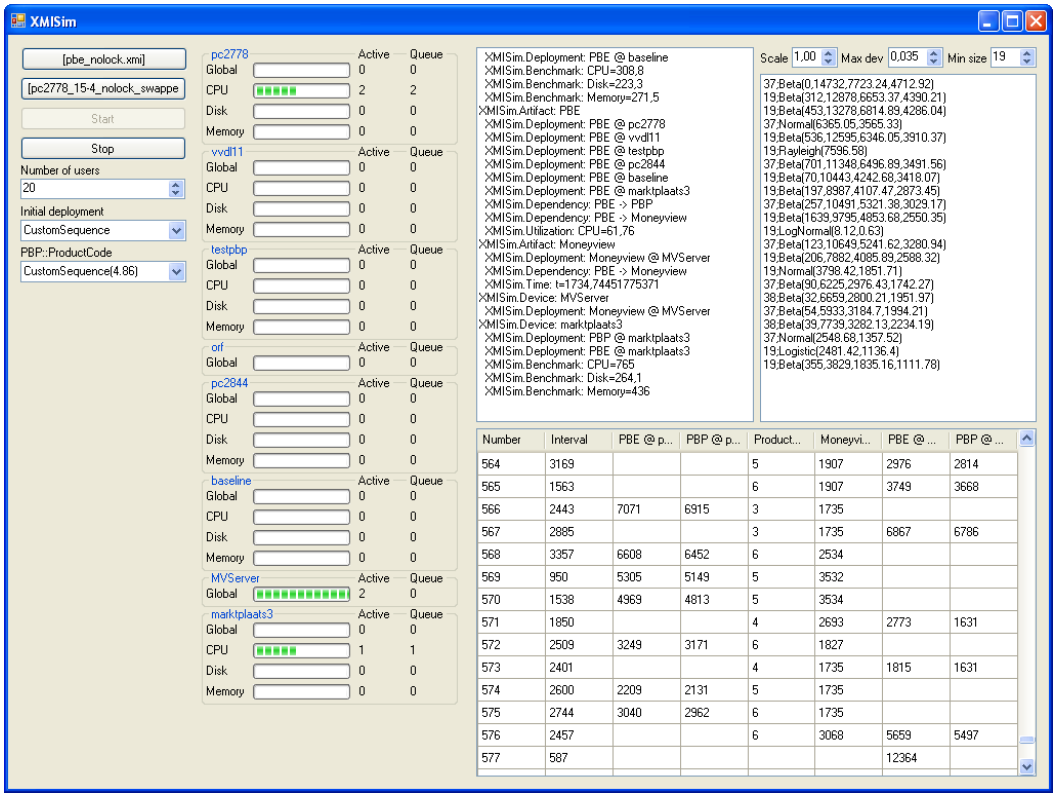

Figure 6: running simulation in XMISim

9 Experimental setup

9.1 Introduction

This section will describe the experimental setup used to determine the correctness of the hardware, software and workload components within SMEPP and the accuracy of the simulator.

9.2 Input components

Three components are needed for simulation. For the hardware component, benchmarks should be run on all systems involved in the environment that needs to be simulated. This is described in Section 5. Similarly, for the software component, each application involved needs to be observed as described in Section 6. All findings should be incorporated in a UML model in the form of constraints connected to devices (systems) and artifacts (applications). Finally the relations among them need to be defined. All of this is described in more detail in Section 7. When the model is finished it should be exported to XMI format.

The third component needed for simulation is the workload description. This can be either a concrete workload in the form of a log file, or a 'synthetic' workload description based on probability distributions. See Section 4.

9.3 Practical validation

When a concrete workload is loaded, the simulation may immediately be started. The simulator will exactly replay the real life workload in this case. By comparing the simulator results with the real life results corresponding to the same workload, the hardware and software model components can be practically validated. This way, it can be made sure that a possible deviation of simulation results from real life results is not due to the workload component within SMEPP.

Instead of exactly replaying the workload from the log file, there is also the option to replay its distributions only. This means that on a long simulation run, the intervals and parameter values will approach those of the real life workload, however the runs won't be exactly the same. When the simulation runs long enough, one may discover scenarios that were not represented in the original workload. Also, by comparing the results of an exact replay with those of a distribution replay, the workload model within SMEPP can be practically validated.

For an exact replay, the simulation results could be compared with the real results by simply comparing the execution times throughout the whole run. However, the results of a distribution replay cannot be directly compared with those of a concrete workload, because the runs are different. In that case the focus should be on the distribution of execution times and their extremes, in order to compare the results.

9.4 Theoretical validation

The experiments described earlier are a practical ‘proof’ of the correctness of different model components and how the simulator handles them. For the workload component, a more theoretical analysis is also possible. A concrete workload is approached by XMISim with a resolution and accuracy set by the user. Naturally, when the resolution is high enough, every workload may be approached with uniform or even constant distributions. However, the workload needs to be modeled with a limited number of distributions (i.e. relatively low resolution) without losing too much detail. Otherwise, the concrete workload might just as well be used for simulation. When decreasing the resolution, it becomes increasingly important that the selected distributions accurately reflect the parts of the original workload they correspond with. Thus, to provide a more theoretical proof of the correctness of the workload component, one should show that the selected distributions are valid in the sense that they reflect the characteristics of the original workload without being too specific or general.

9.5 Prediction

When all SMEPP components are validated, the environment can be used to predict the performance in more abstract situations. Instead of loading a log file and doing an exact replay or distribution replay, the workload parameters may be manually specified. This enables one to predict the results in a case of which a concrete workload example and corresponding results are not available. This is the goal of SMEPP in the end. A combination of both options is also possible. One could load a log file, let XMISim determine its properties and then modify them before starting the simulation. For example, the average request interval could be decreased to simulate the consequences of this modification. Naturally the same holds for changes within the hardware and software components of the system.

10 Experiments

10.1 About Voogd & Voogd

Voogd & Voogd is an excellent example of a company dealing with most of the common problems described in Section 1. The software developed at Voogd & Voogd (V&V from now on) is changing rapidly. Also, the number of users is strongly increasing. Software functionality is tested extensively, however at this moment it is unknown whether non-functional requirements, such as performance, will be satisfied when new software is introduced. In some cases, these requirements are not even known. V&V is also unable to make concrete capacity and availability predictions with respect to the future. This is a typical example of the gap between software engineering and system engineering described earlier. At the moment, all effort to bridge this gap is based on previous experience. This is a critical, however not seldom seen situation. There is a strong need for a more structural approach.

10.2 Applications and servers

PBP (in Dutch: PremieBerekeningsProgramma) is an application developed by V&V that is used by several other applications. Its most important task is to calculate premiums for different products from different insurers, for example to enable comparison of premiums in a consumer specific case. With respect to calculation, PBP can be considered as the heart of the software architecture at V&V.

PBE (in Dutch: Premie Berekening Extern) is also developed internally and can be seen as a wrapper around PBP. A request to PBE is usually forwarded to PBP. In addition to that, PBE also calls an external party called *Moneyview* that essentially does the same job as PBP, however for a different set of insurers. When both are finished, the Moneyview premiums are translated to PBP format and the internal premiums are merged with the external premiums. The result is an enriched list of options for each product. PBE is used by several other applications. The difference mainly lies in the source of the request.

Klik & Sluit is an interface that enables the user to enter criteria (for example the license plate number for a car insurance) after which the concept as described before (PBE, PBP, Moneyview) is used to generate a list of premiums for this case. This form is used by intermediaries, the primary customers of V&V. Intermediary requests are handled by the *Marktplaats* servers. Intermediaries may also offer this functionality to their own customers (mostly consumers) through their website. Consumer requests to Klik & Sluit are handled by the *Diensten* servers. Finally, some of the largest customers of V&V do not send their requests through the Klik & Sluit interface, but use a customized SOAP (Simple Object Access Protocol) interface. These services run on several other servers, however this is beyond the scope of this article.

10.3 Workload analysis example

In this example, the log data of the PBP application running on 4 Marktplaats (M) and 2 Diensten (D) servers within September 2008 is used. Figure 7 and Figure 8 show the request rates on these servers in the complete month (SW: 1 month, TW: 1 month, TR: 1 hour). It is obvious that a pattern repeats itself every week, which means that the target window should be decreased to one week. Also, it seams that all M servers and all D servers show equal behavior, which makes sense because all requests are divided over servers by load balancers.

10.3.1 Decreasing the target window

As said before, when the target window becomes smaller than the source window, there may be multiple values for each data point. In this case for example, the source data from 4 M servers and 4 weeks will lead to 16 values for each data point. Aggregate functions (minimum, maximum, average) will be used to visualize the request rates from now on. Figure 9 and Figure 10 show the request rates on any M or D server in any week of September 2008 (TW is now 1 week). Figure 9 shows peaks during working days and hours, which makes sense because the M servers are used by intermediaries, not by consumers themselves. Figure 10 shows a more equal distribution, as the D servers are used by consumers.

10.3.2 Increasing the source window

Figure 11 and Figure 12 show exactly the same as Figure 9 and Figure 10, except for the fact that all weeks and servers since January 2008 are now included (SW is now 9 months). As you can see the averages are almost equal to the previous graphs, however the minimum and maximum values are, respectively, lower and higher. For example, within the September window, the maximum request rate for an M server was 437 per hour, while for the January-September window, it was 728 per hour. This means that a source window of 1 month is not enough to capture the extremes in this case.

10.3.3 Increasing the resolution

Figure 13 and Figure 14 are the same as Figure 11 and Figure 12, except that the resolution is now 1 minute and the frequencies in the graph are also defined per minute. Remember that the maximum request rate on an M server was 728 per hour, which corresponds to 12.1 per minute. In Figure 13 however, a maximum of 32 requests per minute is shown. This case would not have been discovered without increasing the resolution. Increasing the resolution further to 1 second however, would not make sense because then the average measurement result would be below 1. Recall: for a request rate of 728 per hour, it is worth increasing the resolution to 1 minute, because 728 is larger than 60 (minutes per hour). However, for a rate of 32 per minute, it is not worth increasing the resolution to 1 second, because 32 is smaller than 60 (seconds per minute).

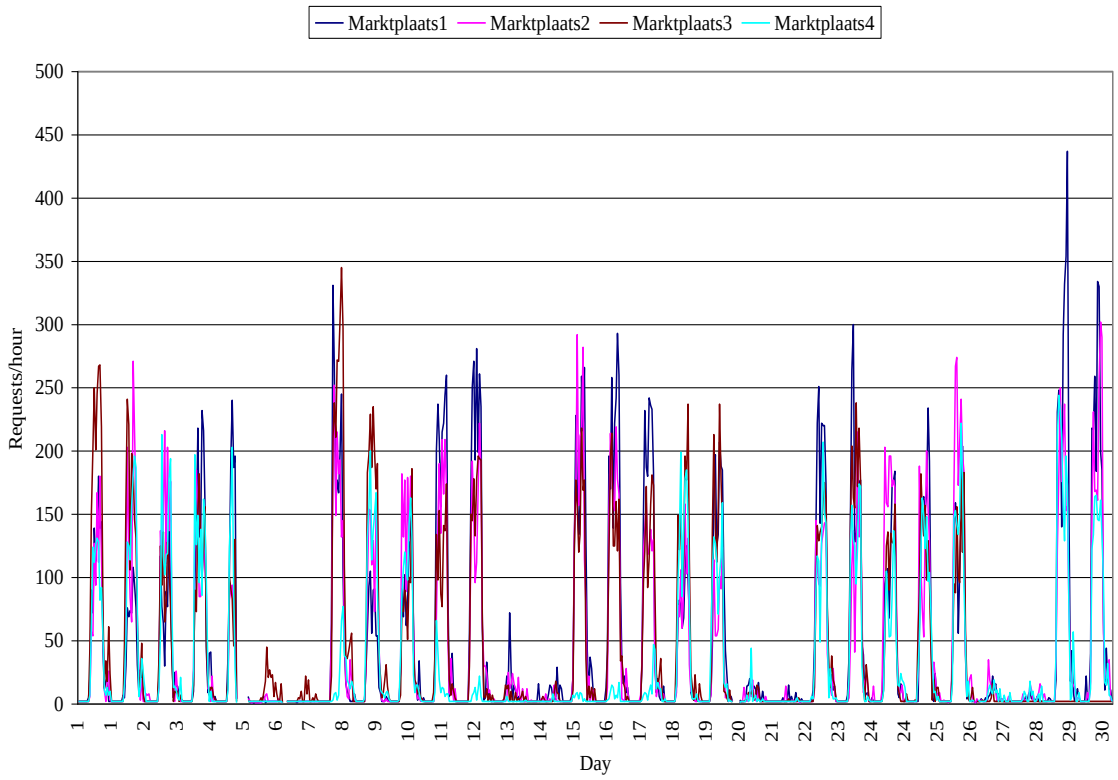

Figure 7: PBP request rate @ Marktplaats (SW: 1 month, TW: 1 month, R: 1 hour)

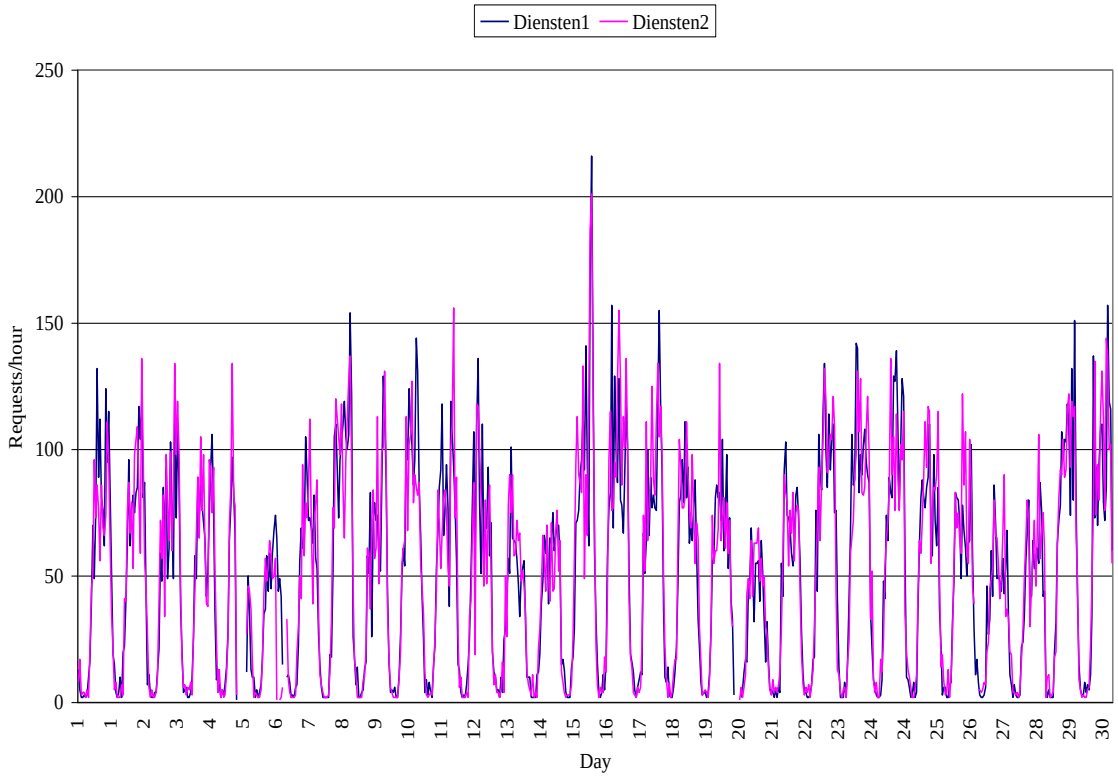

Figure 8: PBP request rate @ Diensten (SW: 1 month, TW: 1 month, R: 1 hour)

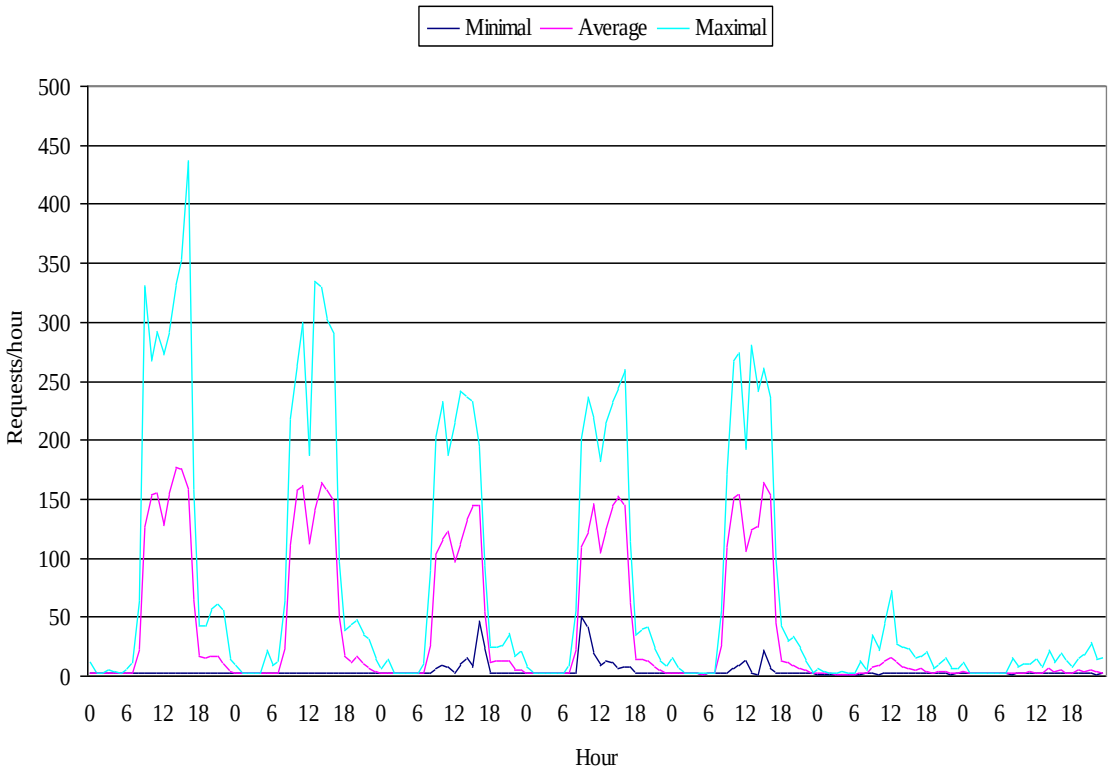

Figure 9: PBP request rate @ Marktplaats (SW: 1 month, TW: 1 week, R: 1 hour)

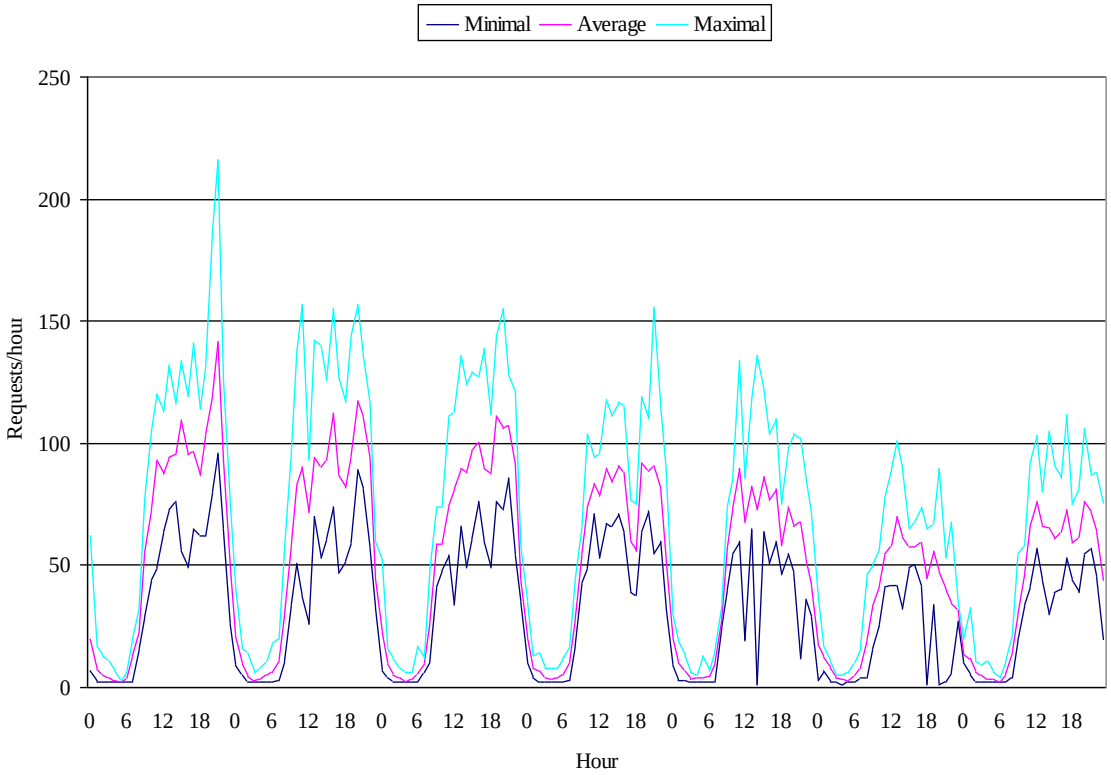

Figure 10: PBP request rate @ Diensten (SW: 1 month, TW: 1 week, R: 1 hour)

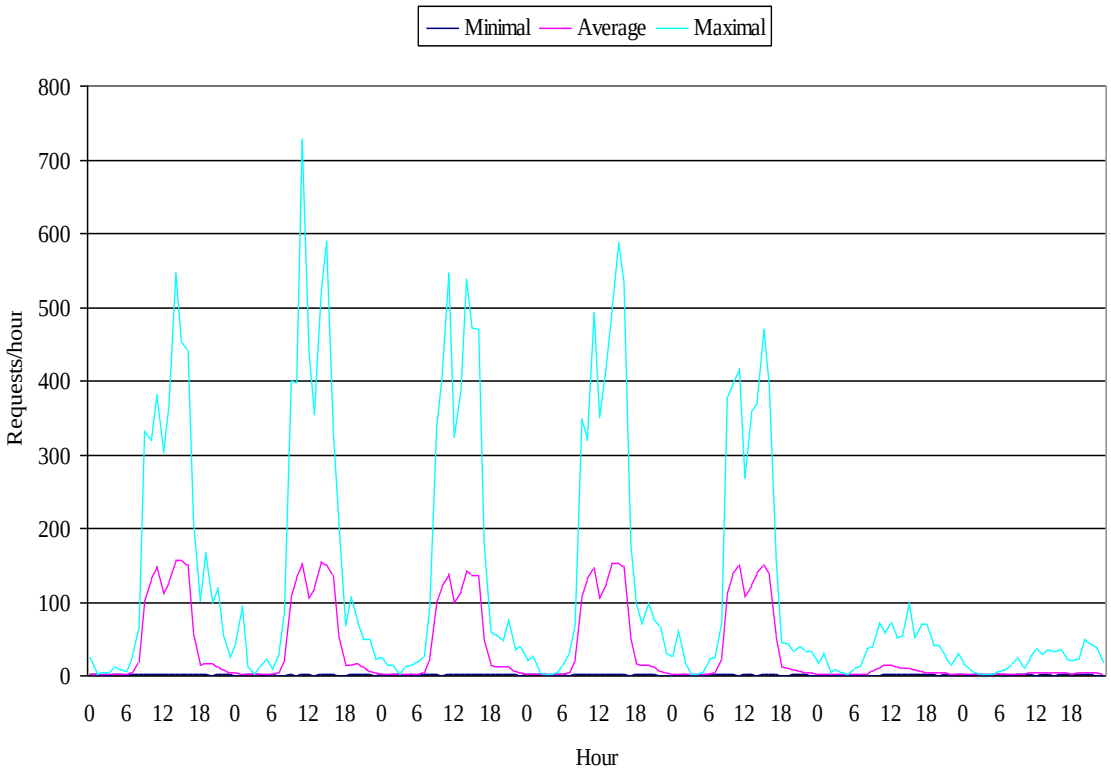

Figure 11: PBP request rate @ Marktplaats (SW: 9 months, TW: 1 week, R: 1 hour)

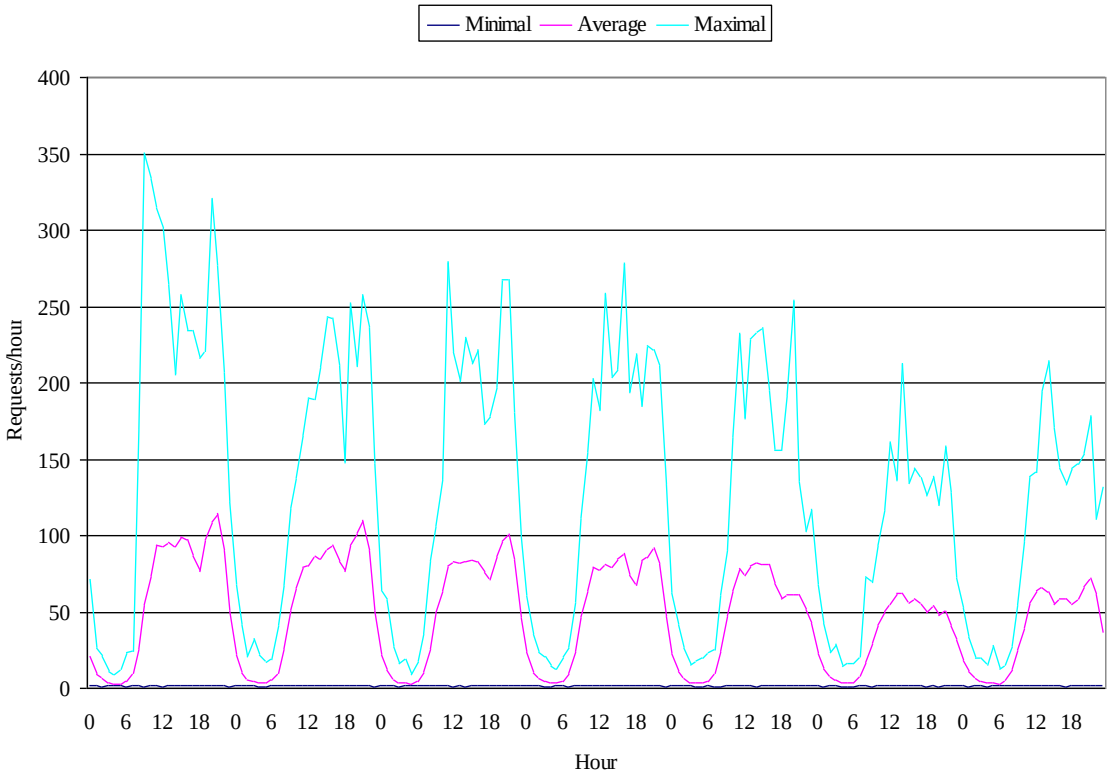

Figure 12: PBP request rate @ Diensten (SW: 9 months, TW: 1 week, R: 1 hour)

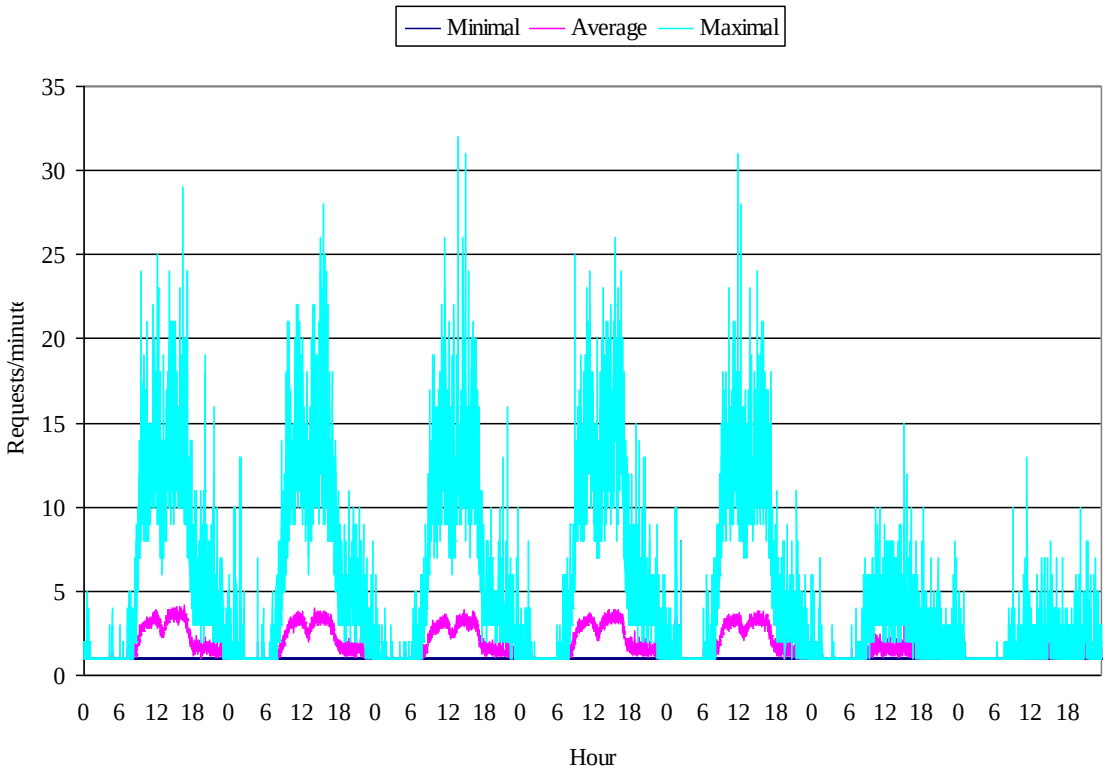

Figure 13: PBP request rate @ Marktplaats (SW: 9 months, TW: 1 week, R: 1 minute)

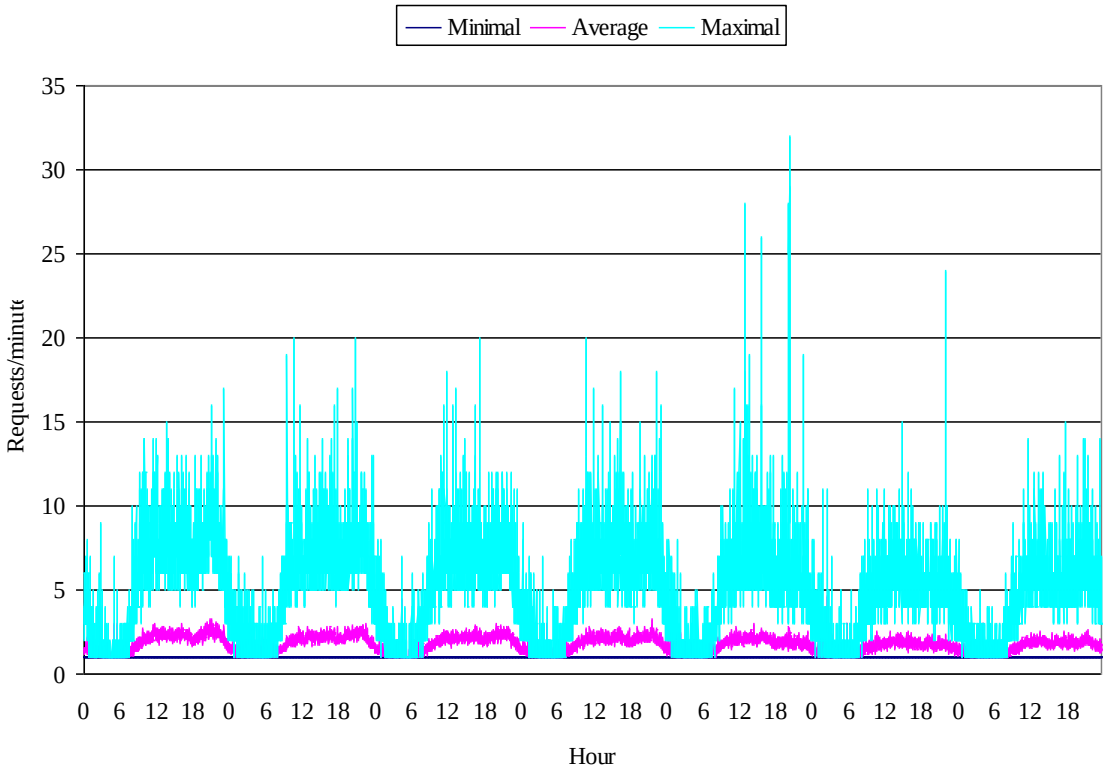

Figure 14: PBP request rate @ Diensten (SW: 9 months, TW: 1 week, R: 1 minute)

10.4 Hardware/software analysis example

In this example the CPU utilization of a certain request to the PBP application is analyzed. As base line a relatively slow single core system with a CPU benchmark value of 308.8 MIPS was used. The system was fully utilized by the PBP request during execution. By running the calculation a large number of times, an optimal execution time of 3199 milliseconds was measured.

Next, exactly the same experiment was done on four other systems with CPU benchmark values ranging from 400 to 1400 MIPS. As described in Section 6.3, the execution time speedup was related to the benchmark speedup (both with respect to the base line system) for each of those systems. This results in the following graph (Figure 15) for the individual efficiency E_i. As you can see the slowest system has an E_i value of 0.67 while the fastest system scores only 0.37. Looking at the optimal execution times, the latter is still significantly faster. However, the individual efficiency (the part of the theoretical speed that is actually used) decreases as the benchmark value increases. This makes sense because the benchmark makes effective use of multiple CPU cores, while the PBP application does not. Remember that for an ideal application, this graph should be a horizontal line at 1, meaning that the theoretical speed is fully utilized and that application speed increases linearly with it. For this example, E_i can be approached by a logarithmic (pink) or power (blue) function:

$$E_i(x) \approx -0.0448 \ln(b_x) + 0.6656$$

$$E_i(x) \approx 0.6662 b_x^{-0.0879}$$

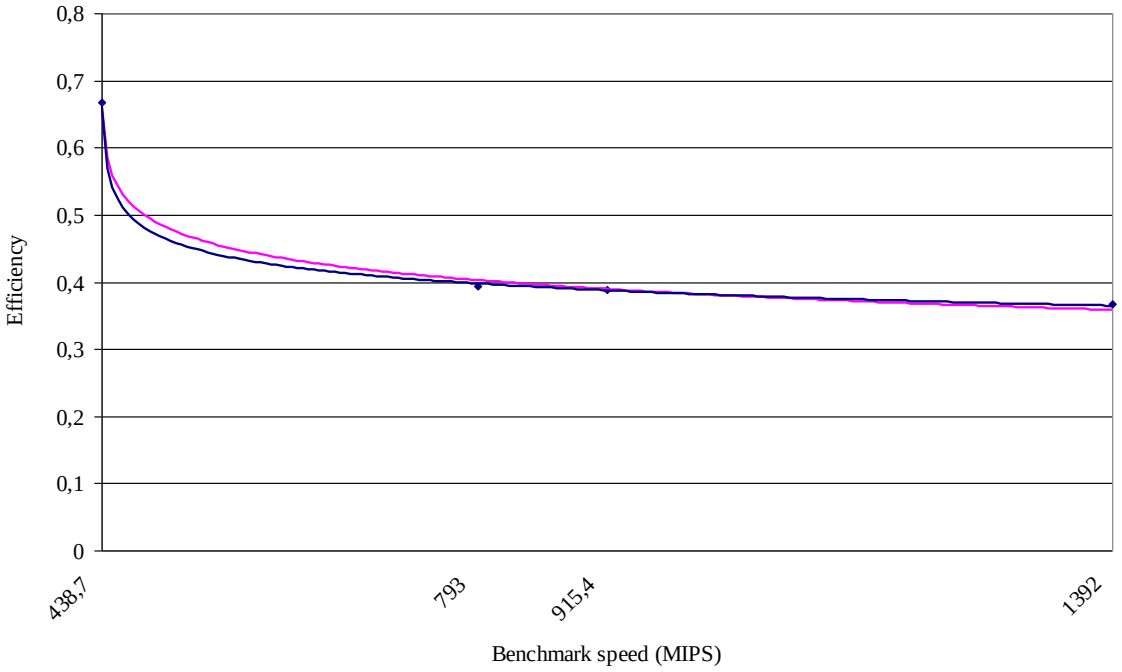

Figure 15: PBP individual efficiency trend

10.5 PBE @ Marktplaats simulation

10.5.1 Hardware and software components

Figure 16 shows a UML model as defined in Section 7 to represent the hardware and software components involved in the PBE case. As you can see, the PBE artifact has two dependencies connecting it to other artifacts: PBP and Moneyview. PBP has four manifestations connecting it to different instances of itself. PBE and PBP have deployments connecting them to the *Marktplaats1* device. The Moneyview artifact is connected to the *MVServer* device through a deployment. PBP has 2 constraints: a parameter called *ProductCode* and an efficiency expression with respect to CPU usage. The instances of PBP each have 3 constraints: a probability of occurrence, a value for the ProductCode parameter and a CPU utilization value. For the Moneyview part, the utilization and benchmark values are unknown, which is why only a time constraint is used.

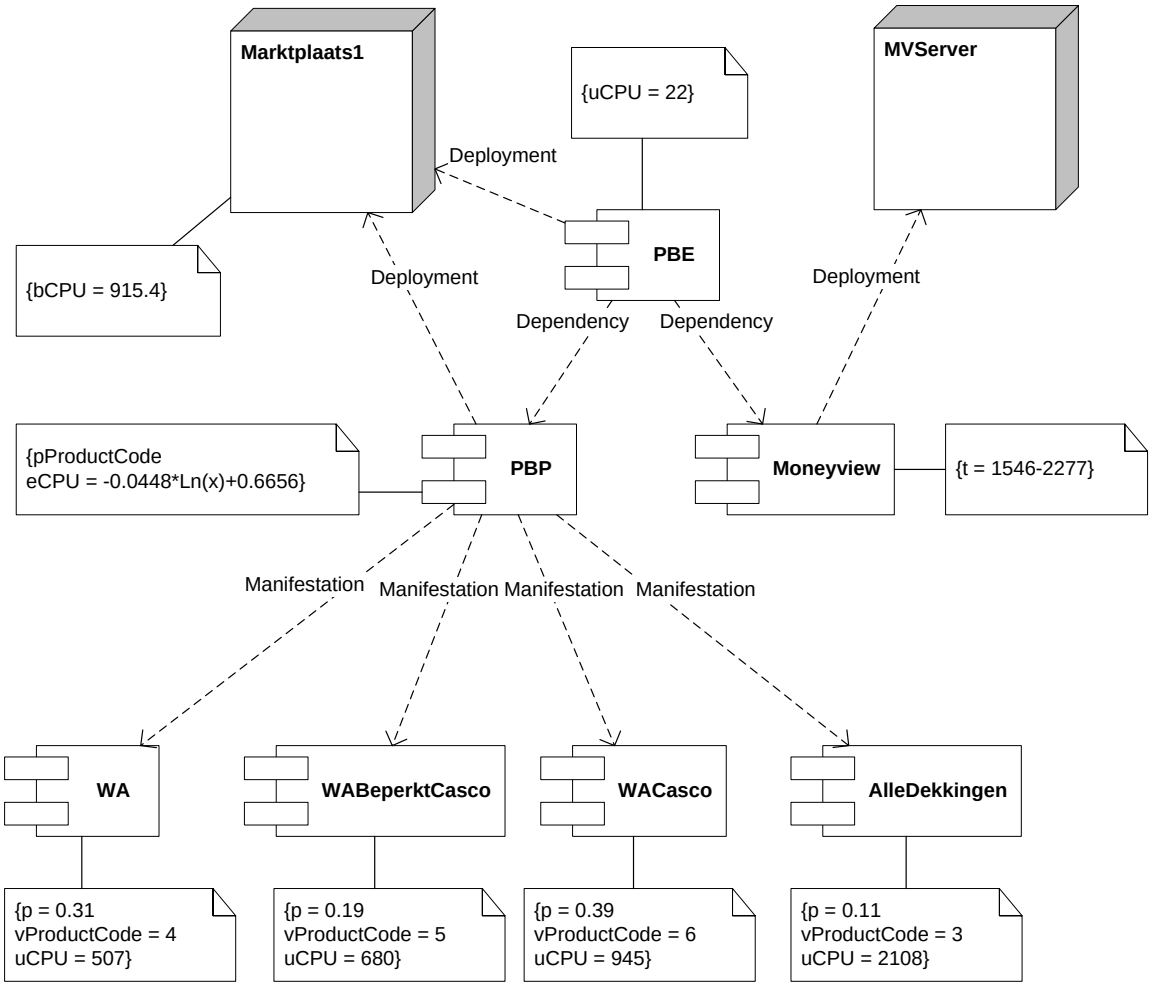

Figure 16: hardware/software model as input for PBE simulation

10.5.2 Practical validation

Figure 17 shows the response times of the PBE application and its dependencies: PBP and Moneyview within a range of 600 requests on an ordinary morning for one of the Marktplaats (M) servers. In general the intervals are decreasing throughout the scenario. As you can see, the PBE time (blue) is determined by the PBP time (pink) in most cases. However it may also be determined by the Moneyview time (yellow). Remember that both dependencies have to be finished before a request to PBE can finish.

When the interval and parameter values of this real life scenario are loaded into XMISim and an exact replay is done, Figure 18 is the result. The PBP graph is very similar to the original graph, which is not surprising because XMISim has simulated exactly the same workload as in real life. The Moneyview graph however is somewhat different. This is because Moneyview is an external party with other customers as well, which means that the Moneyview times cannot be directly related to the workload generated from V&V only. The times are also influenced by other Moneyview customers of whom no data is available, so this part of the scenario cannot be exactly replayed. However the distribution of the real life Moneyview times can be used. This means that the times do not exactly correspond with the real life ones, but over the complete picture, the graphs are comparable, as shown in Figure 18. As a result, the Pearson correlation coefficient of the blue graphs (0.971), though a bit lower than those of the pink (0.987), is very high.

When the log file is loaded into XMISim, the application tries to determine its distributions. Example: with a fixed resolution of 25, the 600 requests in the log file are divided into 24 local parts, each with its own distribution. In this case, XMISim determined two normal, three uniform, one log-normal and one logistic distributed part. The others were best described with a Beta distribution. For the ProductCode parameter, the same probability distribution as in the log file was taken. This means that, during simulation, the parameter values will occur with roughly the same frequencies as in the log file, however the order may be completely different. After running this ‘distribution replay’, Figure 19 is the result. Because the intervals and parameter values used here were only an approach of the real life ones, Figure 19 cannot be directly compared with the previous graphs. However, it is obvious that the characteristics of the graphs are quite similar.

Remember that the request intervals within this scenario were modeled with 24 individual distributions. The real life intervals (and the distributions detected from them) are shown in Figure 20. The resulting intervals that were used for the distribution replay are shown in Figure 21. As you can see, modeling the request intervals like this, and then running a simulation, leads to a comparable however somewhat different scenario. Naturally, using a lower resolution for detecting the distributions will lead to a greater deviation from the original scenario. A workload model like this can be easily modified and as a result, the consequences of such a modification can be easily simulated. Figure 22 shows the intervals that would have been used if XMISim would model the workload with only normal distributions. As you can see that generalization results in a less accurate model at the same resolution.

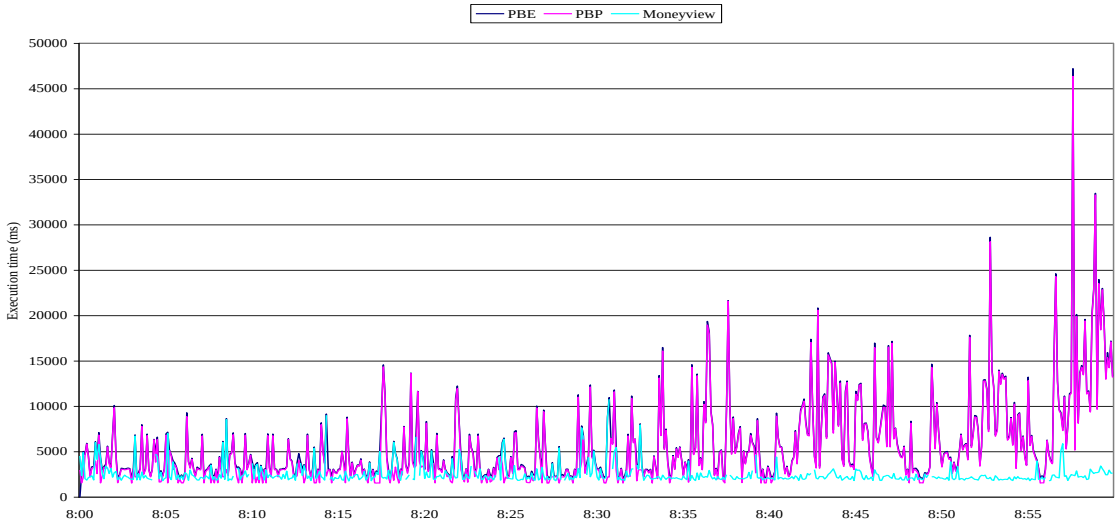

Figure 17: execution times in real life scenario

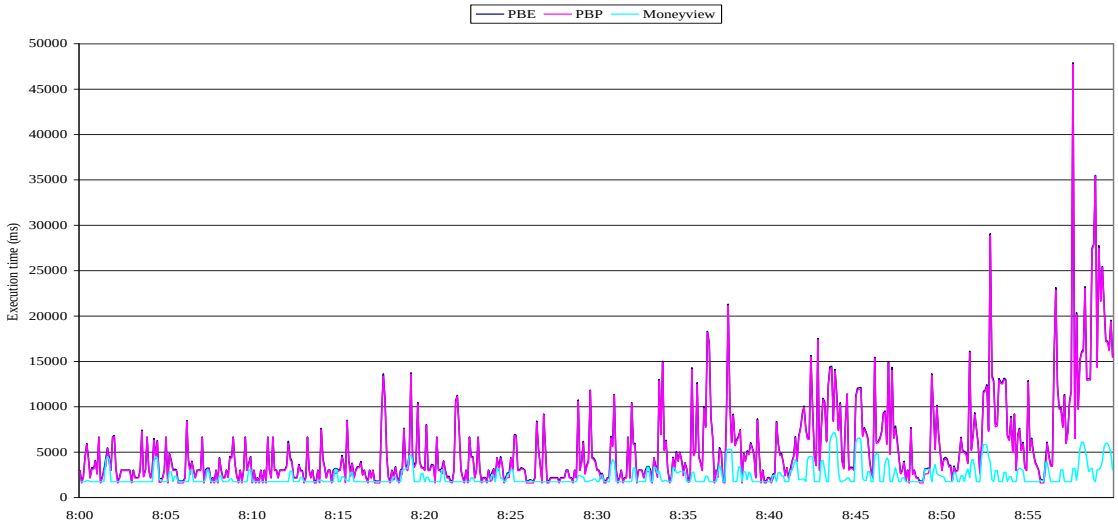

Figure 18: simulated execution times for exact replay

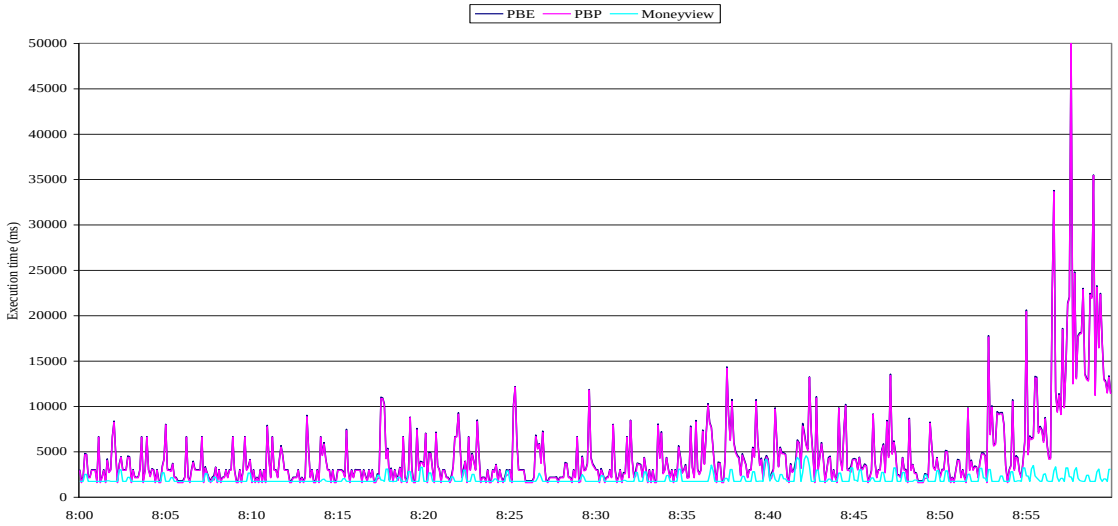

Figure 19: simulated execution times for mixed distribution replay

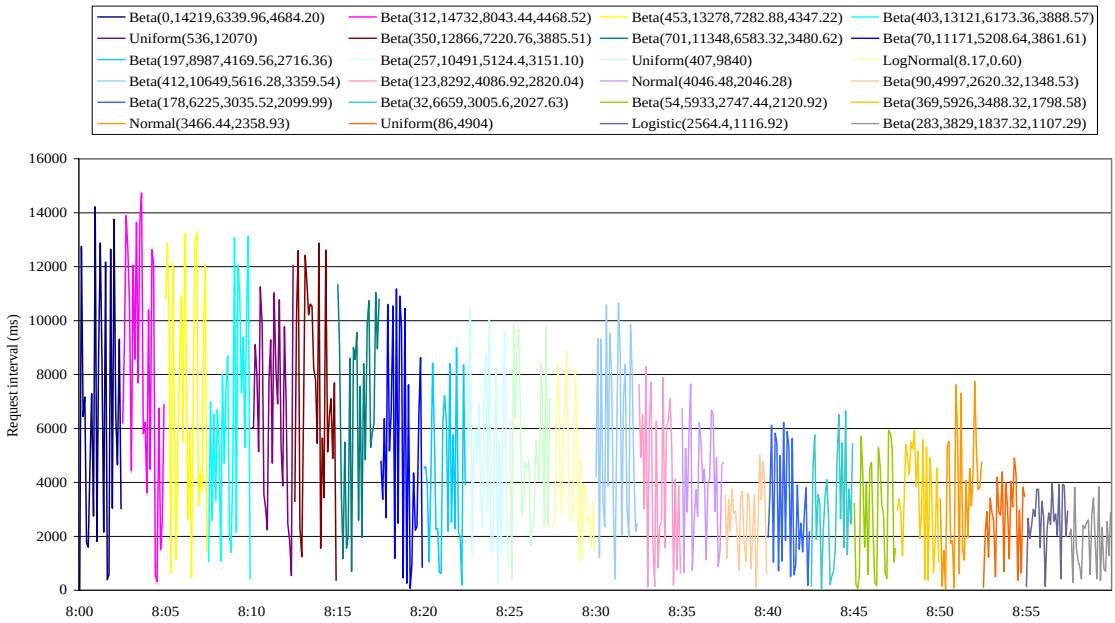

Figure 20: request intervals in real life scenario (Figure 17 and Figure 18)

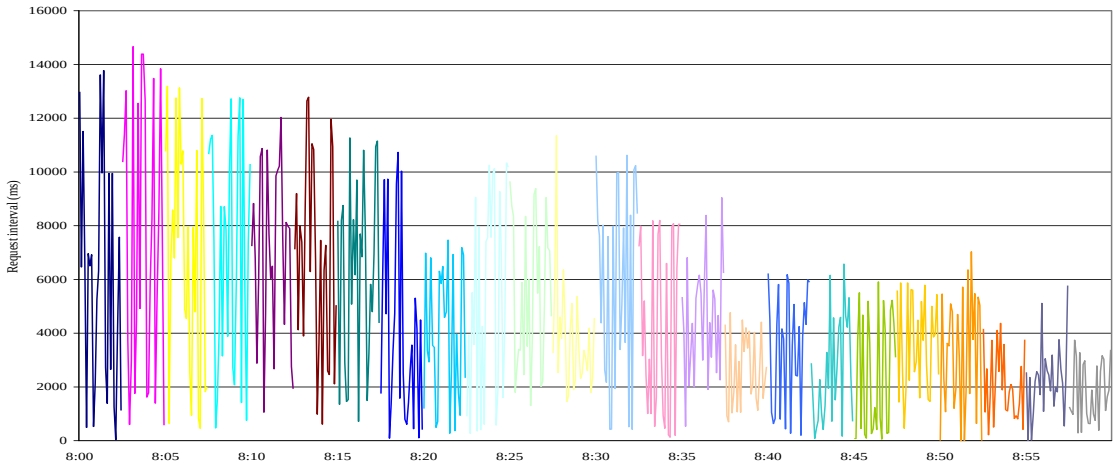

Figure 21: request intervals for mixed distribution replay (Figure 19)

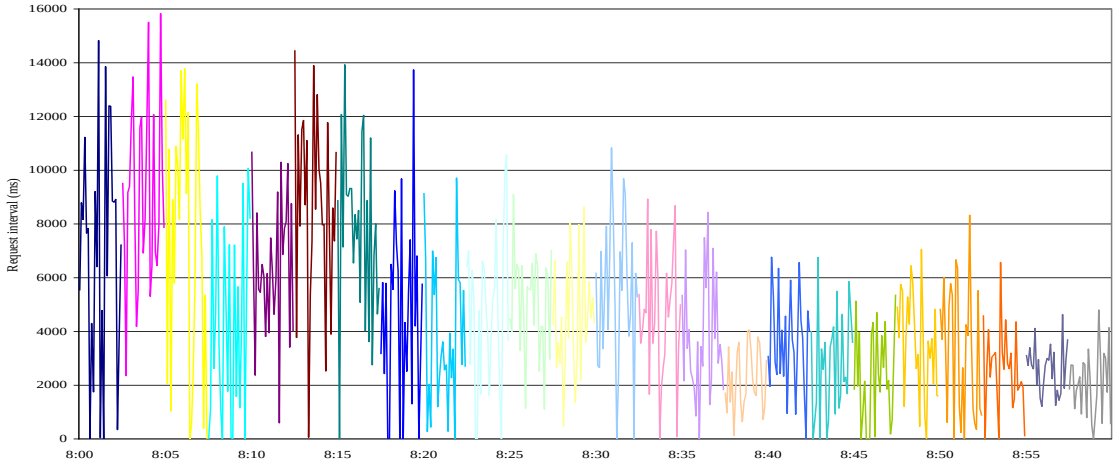

Figure 22: request intervals for normal distribution replay

10.6 PBE critical section case

The previous experiment illustrated the general concept of SMEPP, how to define the workload, hardware and software components and how to simulate the system accordingly. This case will focus on the software component and illustrate how it may be exploited in a decision making process.

As described before, the PBE application depends on both PBP and Moneyview. While the PBE execution times were determined by those of PBP in most cases, the question arose whether it would be more efficient to execute PBP requests within a so-called critical section. In that case, for each instance of the PBE application, only one request to PBP would be processed at any time. Others would have to wait until PBP is available again. As a result, the PBE execution times will be determined by PBP in less cases. In fact, the PBP execution times will all be more or less optimal, because only one request is processed at any time. However, it may also introduce a 'lag' as new requests are posed on PBE while PBP is still processing previous requests. Consequently, PBE may have to wait for PBP and the latter may still be the determining factor for the PBE execution times.

Figure 23 shows the execution times of PBE, again within an ordinary scenario during one hour. The blue line represents real life, while the pink line represents the result of simulating this scenario using the hardware and software models from the previous experiment. Their correlation value of 0.988 is extremely high again. Next, the consequences of introducing a critical section as described above were simulated. To represent this in the software model, the PBP artifact (see Figure 16) was simply supplied with a capacity constraint of value 1. Figure 24 shows the result of simulating the modified model with respect to the scenario before. It also shows the execution times of a PBE version that actually contains the critical section in the same scenario. As you can see, the simulation very accurately predicted the positive consequences (i.e. decreasing execution times) of introducing the critical section. The correlation value of 0.979 was only slightly lower than in the non-critical section case.

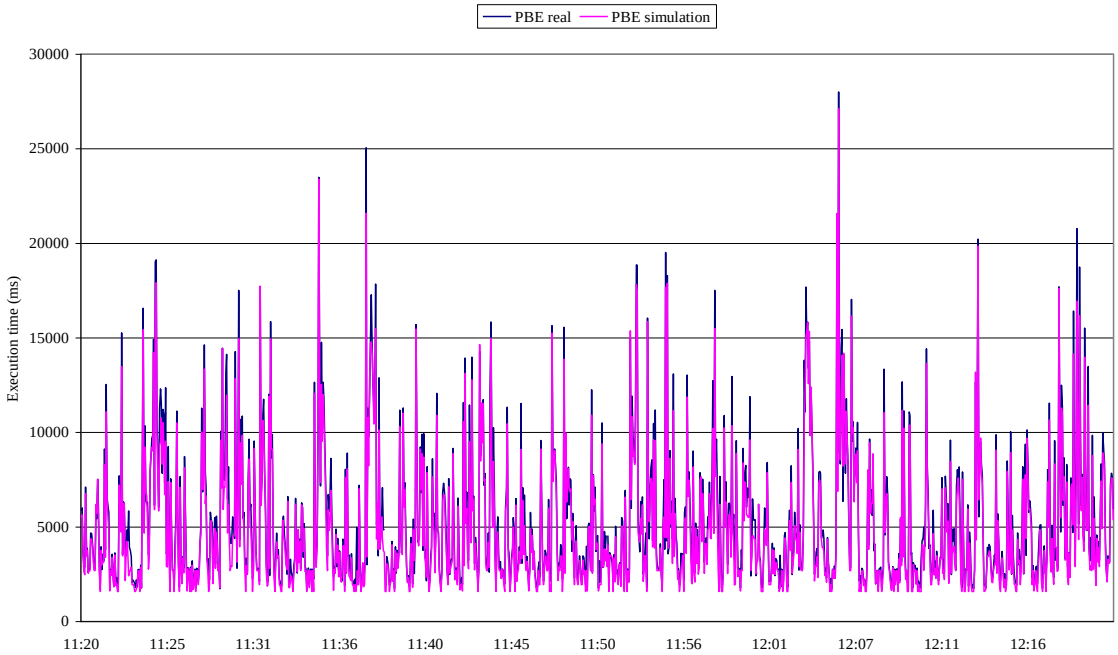

Figure 23: PBE execution times without critical section

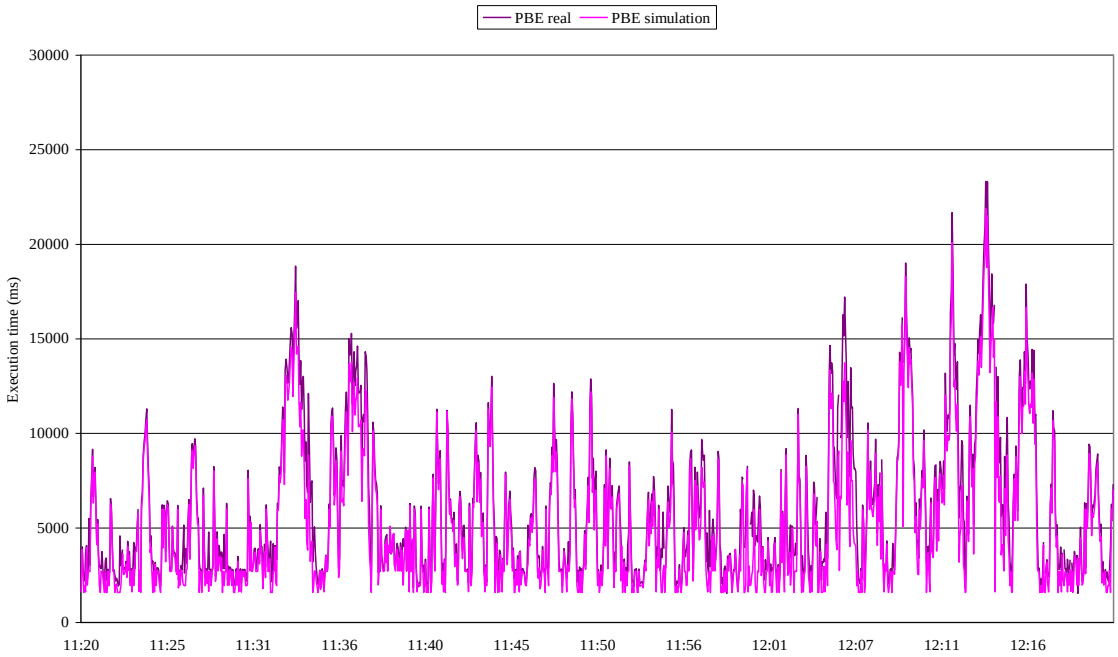

Figure 24: PBE execution times with critical section

10.7 PBP worst case workload scenario

10.7.1 Modeling the worst case scenario

The previous experiments illustrated the concept of SMEPP and its simulation in random scenarios. To validate the workload component within SMEPP in more characteristic scenarios, this section will focus on some extremes with respect to the workload posed on the PBP application. For this task, the worst case scenarios for the past four years (according to the PBP log files) were analyzed. In this respect, a month-to-month trend is hard to discover. Instead, one should compare the intervals within clearly distinct windows, which is why only the first months of each year have been analyzed. Figure 25 shows three examples of worst case scenarios for the PBP application within January and February of 2006, 2007, 2008 and 2009. The periods before and after are also shown in the figures, to make sure that the 'normal' workload is included in the window. In fact, each example shows the intervals within a window of 200 requests with the 'worst case' centered. Note that while the figure shows only three examples per year, the approximations made further on are based on at least 20 worst case scenarios for each year. All of these scenarios have been analyzed using SMEPP. More specifically, XMISim has determined the characteristic parts of each scenario, according to the dynamic optimal resolution algorithm described in Section 4.3. Next, it determined the parameterized distributions that best matched each of these parts. This procedure is also described in Section 4.3. This way, a model of each scenario is automatically generated.

When comparing the example figures, it is clear that the worst case scenarios have become 'worse' through the years. However it appeared that the trends discovered in these scenarios were all more or less similar. This observation is supported by the models generated for each scenario that was analyzed. The 'normal' workload (not to be confused with a normally *distributed* workload), which is shown on the left and right sides of the graphs, was usually characterized by log-normal distributions with parameters μ close to 8.0 and σ between 1.0 and 1.5 (see Appendix B on how to interpret these). The parts in which the intervals either substantially decrease or increase (i.e. increasing or decreasing workload), are best represented by the Gamma distribution, which in fact is a generalization of the exponential distribution. This makes perfect sense, considering that the parameters of these distributions do not restrict their values with a certain upper bound (see Appendix B). In other words: these distributions are best used to represent unpredictable behavior. The center periods, in which the intervals were actually minimal, are best approached by the Beta distribution with parameters α approximately 0.2 and β approximately 0.4. Note that because the Beta distribution is only supported between 0 and 1, these parameters correspond to that support. The result is however rescaled by XMISim to fit the interval distribution as required. The occurrence of Beta distributions in the center periods is explained as follows. When the number of requests in, for example, a single minute increases, the probability of 'extreme' intervals within that minute decreases. Hence, the interval distribution tends to uniformity. Not surprisingly, the uniform distribution is a special case of the Beta distribution. Concluding, a model of the typical worst case workload scenario for the PBP application looks like this (see Section 4.4 for the syntax):

x;LogNormal(μ,σ)
y;Gamma(k,θ)
z;Beta(α,β)
y;Gamma(k,θ)
x;LogNormal(μ,σ)

By using different parameter values, the worst case scenarios of different years can be approached with a single model. Figure 26 shows 3 example scenarios that might occur in each of those years according to the model above. They should be compared to the real life scenarios on the page before. The following parameter values were used to fit the model to the scenarios for the different years.

	x	**y**	**z**	**μ**	**σ**	**k**	**θ**	**α**	**β**
2006	55	35	20	8.49	1.28	0.54	20454.55	0.21	0.38
2007	55	30	30	8.05	1.21	1.05	8704.40	0.21	0.38
2008	55	29	32	8.05	1.20	1.11	3709.76	0.21	0.38
2009	55	35	20	8.04	1.17	0.90	7922.54	0.21	0.38

The Beta distribution parameters α and β, that define the center (i.e. the most busy period) were more or less constant through the years. As a minor generalization, the same values were taken for each year. The most significant evolution is seen in the periods described by the Gamma distributions. Both the interval mean and standard deviation have decreased throughout the years, which is mostly reflected in the scale parameter θ estimated by XMISim. The log-normal parameters μ and σ show a similar, however less significant, evolution. Another interesting development is that the length of the center period *z* has increased, meaning that the period in which the intervals were minimal, lasted longer. However, the last year this effect, as well as the evolution of the Gamma parameters was reversed, meaning that the center period became shorter and the difference between the Gamma and the log-normal periods became smaller. As a result, the intervals over the whole scenario show less variance and the characteristic symmetric decreasing-and-then-increasing shape slowly disappears.

10.7.2 Simulating the worst case scenario

Now that a worst case scenario model with parameters for each year is available, the execution times of the PBP application (with respect to this scenario) can be simulated and compared with the real life times. Remember that the PBP execution times are only partly determined by the request intervals. A large part is also determined by the values of the ProductCode parameter (see Figure 16). To remove this 'noise', the execution times need to be normalized for each different ProductCode value. As a result, a low execution time will be close to zero and a high execution time will be close to one, regardless of the ProductCode value.

For each of the worst case scenarios in Figure 25 and Figure 26 the corresponding (normalized) execution times are also shown at the top of the graph. Note that the values

on the vertical axis correspond to the intervals, not the execution times. Remember that Figure 25 shows real life execution times according to the scenarios from 2006 to 2009, while Figure 26 shows simulated execution times according to a single workload model, which is parameterized to fit the different years. Hence, if this model is correct, the execution times in these figures should be globally comparable. However the actual order and peak locations may be completely different. Thus, one should not directly compare these lines. Instead, one should compare their characteristics. For example, the mean execution time in the simulated 2009 scenarios is 454 ms, while in the real scenarios it is 380 ms. A more sophisticated comparison uses the distributions of execution times in the real life and simulated figures, as demonstrated in Figure 27. The blue columns represent the frequencies of execution times in ranges of 100 ms. The pink line shows the cumulative probability of execution times, i.e. the probability that the execution time is smaller than or equal to the value on the horizontal axis, within the scenarios represented.

Although the cumulative probabilities correlate for nearly 0.99, the simulated execution times in Figure 27 show less variance than the real ones. This is also characterized by the 'jumps' in the cumulative probability lines. These are due to some generalizations and assumptions in the models of the PBP application and the workload posed on it. More specifically:

- The workload model was parameterized to evolve in a way comparable to real life observations. However essentially the model was the same for all years.
- The application model includes the optimal execution times for 6 of the ProductCode values where they were most distinct, while at least 16 different values occurred throughout the scenarios. As a result, requests with other ProductCode values will have approximately equal execution times, provided that only one of them is executed simultaneously.
- The optimal execution times that were modeled are based on 2008 measurements. Any evolution of the application itself was not modeled.

Keeping all this in mind while looking at Figure 27, the simulated execution times are actually very accurate. To say the least, the simulation provides a good impression of the execution times in real life scenarios. It is quite surprising that the performance of a heavy weight application like PBP, under a worst case workload, can be predicted on this level with models as abstract as those used here.

10.7.3 Correctness

When comparing the request intervals and the normalized execution times in Figure 25 and Figure 26, it becomes clear that low intervals correspond to higher frequencies of high execution times, in both the real and simulated scenarios. However, in real life this correspondence has become less obvious through the years, which can be explained because at the same time, the interval scenarios have become more and more 'flat'. Naturally the dependency of execution times on intervals decreases when the variance in the latter decreases. The opposite relation, intervals depending on execution times, may also play a role. For each interval throughout the scenario, XMISim has determined the

probability that it was depending on the execution time of any running request as described in Section 4.5. This probability was 0.04 at most, perhaps explained by the fact that in the 2009 scenarios the average number of requests per user was only 1.5. Thus, within this window of 200 requests, at most 8 (and probably less) of the intervals have been partly determined by execution times. Although in these scenarios the dependency of intervals on execution times is small, XMISim does take it into account upon simulation.

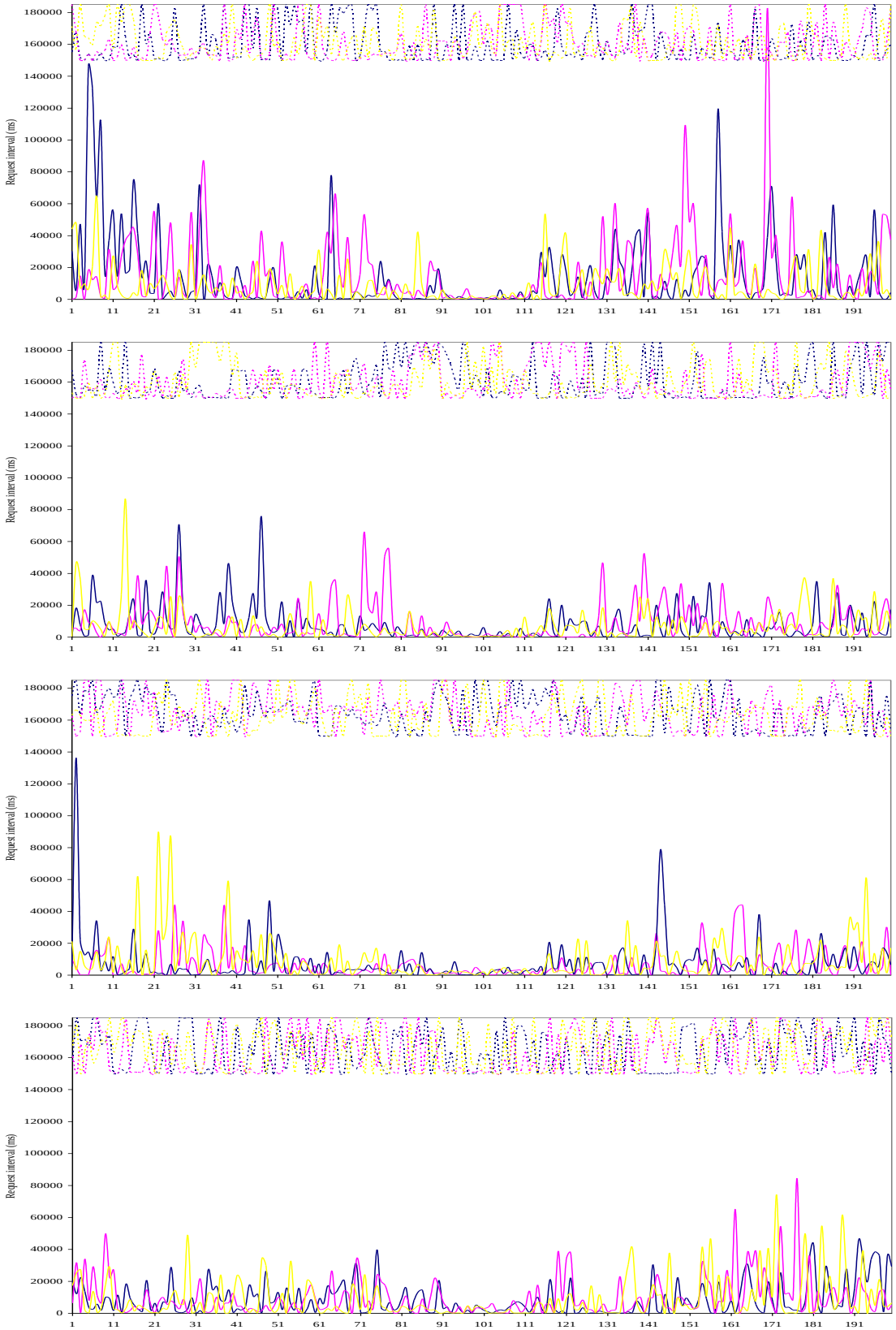

Figure 25: top 3 worst case scenarios in January/February 2006/2007/2008/2009

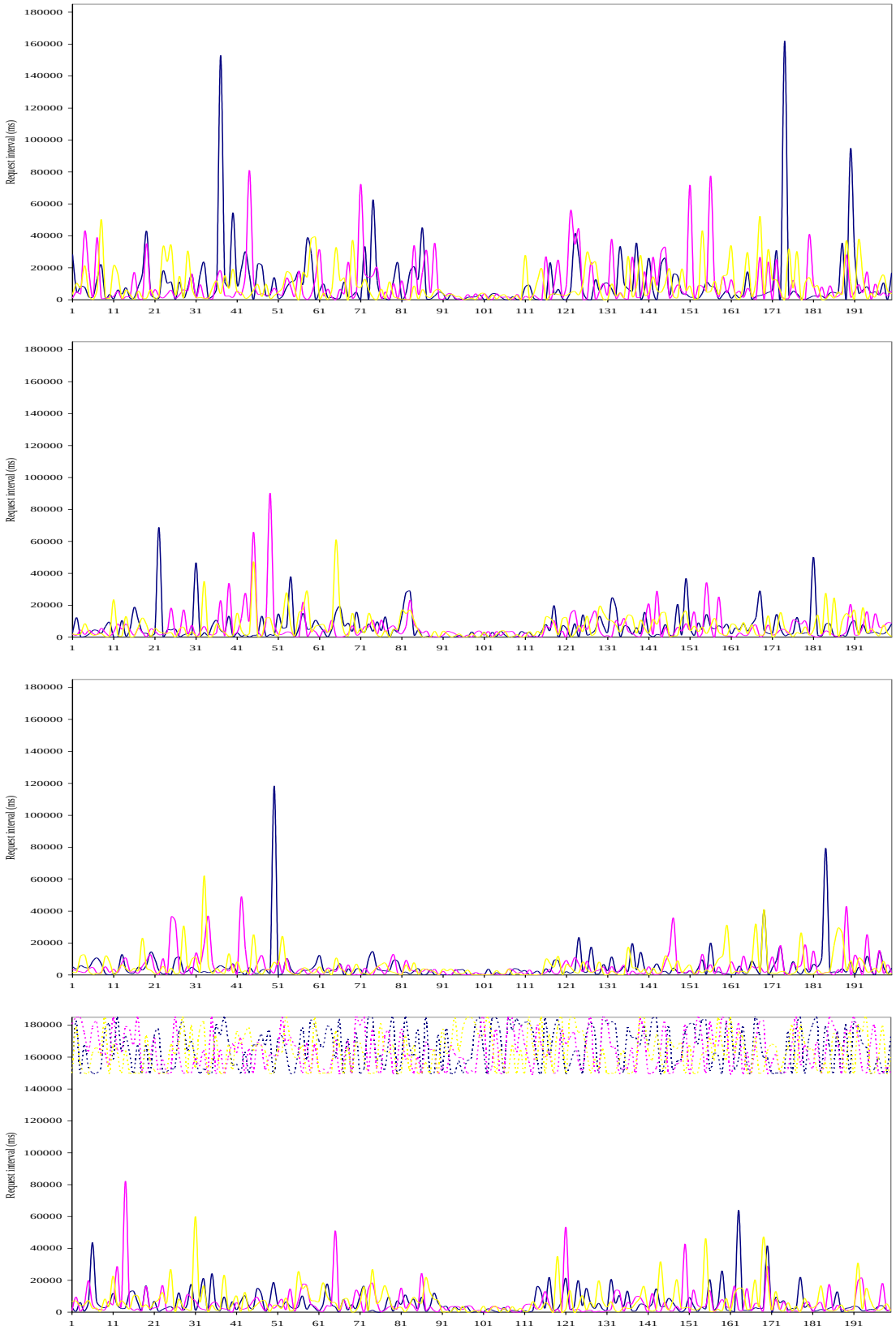

Figure 26: example scenarios with 2006/2007/2008/2009 parameters

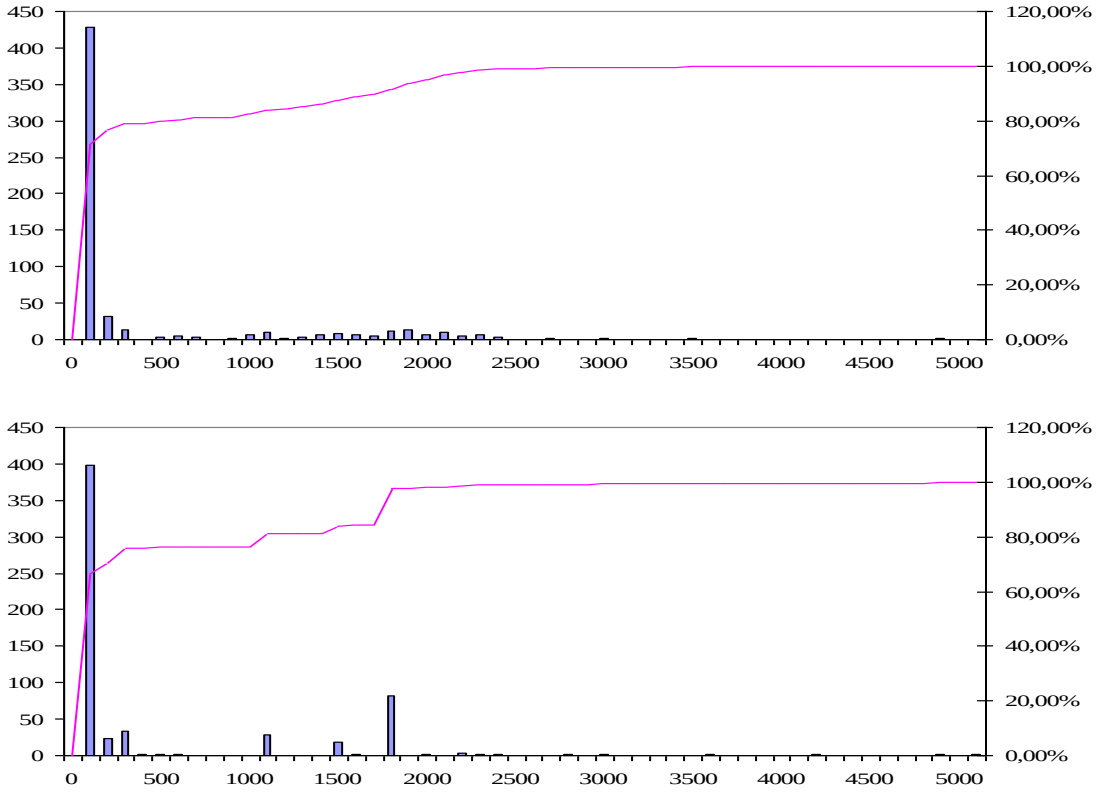

Figure 27: distribution of execution times in real and simulated 2009 scenarios

11 Conclusions

This article introduces SMEPP, a novel System Modeling Environment for Performance Prediction. The environment consists of methods for modeling the entire context of a system. Additionally, it includes an application to simulate these models. The simulator, XMISim, needs two types of input. First, it expects a model of the hardware and software components within the system concerned. The hardware/software model used within SMEPP is an extended version of the UML deployment diagram. It is a flexible model in which many real life systems [45] may be represented. See Section 7 for details. A hardware/software model can be composed in any graphical UML environment that includes functionality to export the model into the standard XMI format. As its name suggests, XMISim expects the model to be in this format.

In addition to the hardware/software model, XMISim expects a workload model. This is a description of the requests that should be posed on the system during simulation. At first, extensive work has been done to make sure that the simulator produces valid results with respect to concrete workloads. In other words: real life scenarios (as extracted from log files and databases) were exactly replayed in order to compare the simulator results with the real life ones. Here, 'results' refers to the execution times of the modeled system and its components. Correlation values no lower than 0.97 were seen in these cases. Examples are given in Section 10.5 and 10.6.

Naturally, exactly replaying a real life workload scenario is useful for validation of the hardware/software model. However for predictive purposes, which is the main goal, a generalized representation of such a scenario was needed. Therefore, the next focus was on modeling workload scenarios. This has resulted in a method of determining the characteristic parts within a concrete workload and for each of these parts, finding the best approximation among a set of probability distributions. See Section 4.3 for details. XMISim was extended with functionality to semi-automatically execute this process. As a result, the application can generate a workload scenario model by itself. This model may directly be used in a subsequent simulation. Using a workload model instead of a concrete workload has several advantages, as described in Section 4.4.

SMEPP enables one to quickly and accurately assess the performance of a (proposed) system based on a minimal amount of input information. Also, the consequences of updates within the hardware and software components and the evolution of workload can be evaluated. Section 10.7 gives an example of how extremely abstract hardware/software and workload models can be simulated to provide a quick and valuable insight in the performance of the system concerned. In this respect, the goal of requiring a minimal amount of input information was accomplished.

One of the other goals, the use of existing methods, was partly accomplished. For the modeling and analysis parts, lots of techniques already known in literature were combined. Only for the simulation part a new solution has been developed, because existing simulators were focused on either hardware or software models (both were

needed), while the ability to accurately simulate a workload scenario was insufficient or even absent.

Particularly because of its wide context, which was the third goal, SMEPP is applicable in real life situations. This is illustrated by various cases, aimed at both validation and prediction. It is shown that SMEPP is a valid, scalable and general approach to system modeling and performance prediction.

12 Future work

This article has presented methods of retrieving information about a system, modeling the system and finally simulating this model. Although the whole process fits together quite nicely, some improvements might make life easier. In addition to that, some suggestions are given to enlarge the area of applicability for SMEPP.

It was described how to define existing hardware and software components, as well as concrete workloads in models suitable for simulation. Additionally, it was shown how the evolution in any of these components may be exploited to approach the results in more abstract cases. For example, given the evolution of workload within the past years, the workload within the next year may be approached. However, to represent *any* given case (possibly without any real life information) still requires quite some expertise. For example, if one is asked to simulate a 40 % increase in the current 'average' workload, it is quite a challenge to translate such a rough metric into a scenario which is both representative and suitable for simulation. In practice however, this is the type of input that should be expected. Hence, it would be interesting to see if this process can be made any easier by developing methods that automatically perform these kinds of translations.

Section 6.3 describes a method of relating measured execution times of an application to the systems benchmark values, in order to discover an individual efficiency trend E_i. For this task curve fitting techniques in external statistical tools are used. The resulting expression is then included in the UML model, to be evaluated during simulation. Instead, it would be nice to integrate the curve fitting process in XMISim. This way, one would simply put the measured execution times and benchmark values in the model and their relation would be determined on the fly, without the need for external tools.

The output of XMISim is currently restricted to a list of execution times for the system and its components, accompanied by some basic statistics. However, to perform a thorough analysis of the results external tools are still needed. An example is determining the distribution of execution times as demonstrated in Section 10.7. It would be useful to integrate this process, in order for XMISim to automatically provide such an analysis after simulation.

The simulation experiments have focused on CPU and memory performance, as these were most determining for the performance of the applications concerned. More specifically, the benchmarks that were used, as well as the process of relating them to execution times, were aimed at these resources. Although the framework that was presented might just as easily be used to simulate, for example, graphics performance, it is currently unknown how accurate the results would be.

Finally, it would be interesting to evaluate the applicability of SMEPP and the accuracy of XMISim with respect to virtual systems.

13 References

1. Connie U. Smith and Lloyd G. Williams, “Performance and Scalability of Distributed Software Architectures: An SPE Approach”, Parallel and Distributed Computing Practices, 2002.
2. R. Scott Barber, Beyond performance testing, IBM DeveloperWorks, 2004.
3. R.J.A. Buhr and R.S. Casselman, Use Case Maps for Object-Oriented Systems, Prentice Hall, 1996.
4. Object Management Group, Unified Modeling Language (UML) specification, Version 2.1.2, November 2007.
5. Object Management Group, UML Profile for Schedulability, Performance and Time specification, Version 1.1, January 2005.
6. Object Management Group, UML Profile for Modeling QoS and Fault Tolerance Characteristics and Mechanisms specification, Version 1.1, April 2008.
7. Object Management Group, UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE) specification, Version 1.0 Beta 2, June 2008.
8. Object Management Group, Systems Modeling Language (SysML) specification, Version 1.0, September 2007.
9. Simona Bernardi and Dorina C. Petriu, “Comparing two UML Profiles for Non-functional Requirement Annotations”, the SPT and QoS Profiles, UML'2004, 2004.
10. Peter King and Rob Pooley, “Derivation of Petri net Performance Models from UML Specifications of Communications Software”, Proc. XV UK Performance Engineering Workshop, pp 262-276, 2000.
11. Juan Pablo Lopez-Grao, José Merseguer and Javier Campos, “From UML Activity Diagrams To Stochastic Petri Nets”, Proc. 4th Int. Workshop on Software and Performance (WOSP 2004), pp 25-36, January 2004.
12. Murray Woodside, Dorina C. Petriu, Dorin B. Petriu, Hui Shen, Toqeer Israr and José Merseguer, “Performance by Unified Model Analysis (PUMA)”, Proc. 5th Int. Workshop on Software and Performance (WOSP 2005), pp 1-12, July 2005.
13. Murray Woodside, Greg Franks and Dorina C. Petriu, “The Future of Software Performance Engineering”, Proc. Future of Software Engineering 2007, IEEE Computer Society Order Number P2829, pp 171-187, May 2007.
14. Connie U. Smith and C.M. Llado, “Performance Model Interchange Format (PMIF 2.0): XML Definition and Implementation”, QEST '04: Proc. Quantitative Evaluation of Systems, First International Conference, pp 38-47, 2004.
15. Object Management Group, MOF 2.0/XMI Mapping, Version 2.1.1, December 2007.
16. R. Scott Barber, “Creating Effective Load Models for Performance Testing with Incomplete Empirical Data”, Proc. 6th IEEE Int. Workshop on Web Site Evolution, pp 51-59, 2004.
17. Alberto Savoia, “Web Load Test Planning: Predicting how your Web site will respond to stress”, STQE Magazine, pp 32-37, March/April 2001

18. Allen D. Malony and B. Robert Helm, “A theory and architecture for automating performance diagnosis”, Future Generation Computer Systems 18(1), pp 189-200, September 2001.
19. Marc Lankhorst, Enterprise Architecture at Work: Modelling, Communication and Analysis, Springer, 2005.
20. John D. McCalpin, “Memory Bandwidth and Machine Balance in Current High Performance Computers”, IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter, December 1995
21. Reinhold P. Weicker, “Dhrystone: a synthetic systems programming benchmark”, Communications of the ACM Volume 27, pp 1013-1030, October 1984.
22. Harold J. Curnow and Brian A. Wichman, “A Synthetic Benchmark”, Computer Journal Volume 19 Issue 1, pp 43-49, February 1976.
23. Ian H. Witten, Radford M. Neal and John G. Cleary, “Arithmetic Coding for Data Compression”, Communications of the ACM Volume 30 Issue 6, pp 520-540, June 1987.
24. C.A.R. Hoare, “Partition: Algorithm 63, Quicksort: Algorithm 64 and Find: Algorithm 65”, Communications of the ACM Volume 4 Issue 7, pp 321-322, 1961.
25. Bruce Schneier, “Description of a New Variable-Length Key, 64-bit Block Cipher (Blowfish)”, Fast Software Encryption, pp 191-204, 1993.
26. Werner Almesberger, “UML Simulator”, Ottawa Linux Symposium, July 2003.
27. Andrei Kirshin, Dolev Dotan and Alan Hartman, “A UML Simulator Based on a Generic Model Execution Engine”, IBM Haifa Research Lab, 2006.
28. Maria Victoria Cengarle, Juergen Dingel, Hans Gronniger and Bernhard Rumpe, “System-Model-based Simulation of UML Models”, Technische Universitat Munchen, August 2007.
29. Frank de Boer, Marcello Bonsangue, Hugo ter Doest, Luuk Groenewegen, Henk Jonkers, Andries Stam and Leon van der Torre, “Analysis of Enterprise Architectures”, CWI/LIACS/Ordina/TI, June 2003.
30. Maria-Eugenia Iacob and Henk Jonkers, “Quantitative Analysis of Enterprise Architectures”, Telematica Instituut (TI), March 2004.
31. Tim Weilkiens, “Systems Engineering with SysML/UML: Modeling, Analysis, Design”, Morgan Kaufmann OMG Press, 2006.
32. Matthew Hause, “The SysML Modelling Language”, Fifth European Systems Engineering Conference, September 2006.
33. Matthew Hause and Francis Thom, “Building Bridges Between Systems and Software with SysML and UML”, INCOSE, 2008.
34. Moreno Marzolla and Simonetta Balsamo, “UML-PSI: the UML Performance Simulator”, Proceedings of The Quantitative Evaluation of Systems, First International Conference, pp 340-341, 2004.
35. Moreno Marzolla, “libcppsim: A Simula-like, portable process-oriented simulation library in C++”, 2004.
36. Jing Xu, Murray Woodside, Dorina Petriu, “Performance Analysis of a Software Design using the UML Profile for Schedulability, Performance and Time”, Proceedings TOOLS 03, 2003.

37. Andrew J. Bennett and A.J. Field, “Performance Engineering with the UML Profile for Schedulability, Performance and Time: a Case Study”, Proceedings MASCOTS 2004, pp 67-76, 2004.
38. Theodore Wilbur Anderson and Donald A. Darling, “Asymptotic theory of certain “goodness of fit” criteria based on stochastic processes”, Annals of Mathematical Statistics 23, pp 193-212, 1952.
39. M.A. Stephens, “EDF Statistics for Goodness of Fit and Some Comparisons”, Journal of the American Statistical Association Volume 69, pp 730-737, 1974.
40. Joseph Lee Rodgers and W. Alan Nicewander, “Thirteen Ways to Look at the Correlation Coefficient”, The American Statistician Volume 42 Number 1, pp 59-66, February 1988.
41. Jérémie Gallien, “Common Probability Distributions for Simulation Modeling”, MIT Sloan School of Management, October 2003.
42. John A. Rice, “Mathematical Statistics and Data Analysis, Second Edition”, DuxBury Press, June 1994.
43. Michael P. McLaughlin, “A Compendium of Common Probability Distributions, Third Edition”, September 1999.
44. Johnson, Kotz and Balakrishnan, “Continuous Univariate Distributions, Volumes 1/2, Second Edition”, John Wiley and Sons, 1994.
45. Prasad Jogalekar and Murray Woodside, “Evaluating the Scalability of Distributed Systems”, IEEE Transactions on Parallel and Distributed Systems Volume 11 Number 6, pp 589-603, June 2000.
46. C. Cavenet, S. Gilmore, J. Hillston, L. Kloul, and P. Stevens, “Analysing UML 2.0 activity diagrams in the software performance engineering process”, Proc. 4th Int. Workshop on Software and Performance (WOSP 2004), pp 74-83, January 2004.
47. Simonetta Balsamo and Moreno Marzolla, “Simulation Modeling of UML Software Architectures”, Proc. ESM'03, June 2003.

Appendix A: XMISim class diagram

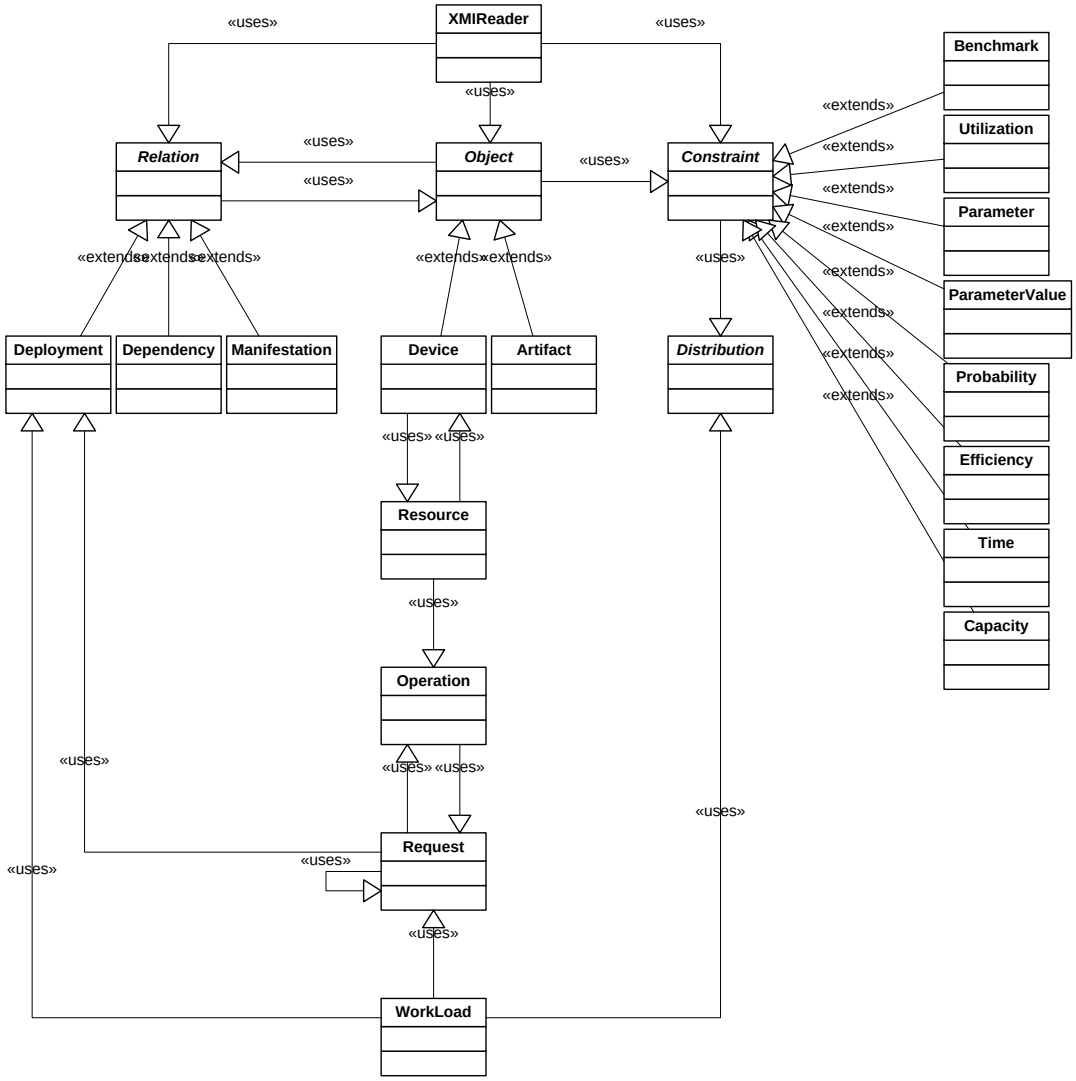

Figure 28: Class diagram for XMISim

Appendix B: continuous probability distributions and their cumulative distribution functions

Exponential distribution

$$F(x;\lambda) = 1 - e^{-\lambda x}, \quad x \geq 0$$
$$\lambda = 1/mean \qquad [44]$$

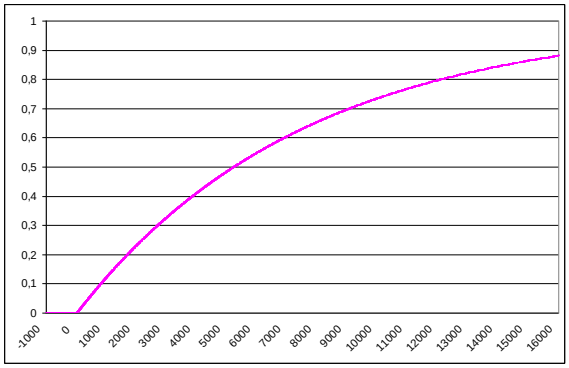

Figure 29: Exponential F(x; 1/7500) such that mean = 7500

Normal distribution

$$F(x;\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{x} \exp\left(-\frac{(t-\mu)^2}{2\sigma^2}\right) dt$$
$$\mu = mean, \sigma = stdev$$

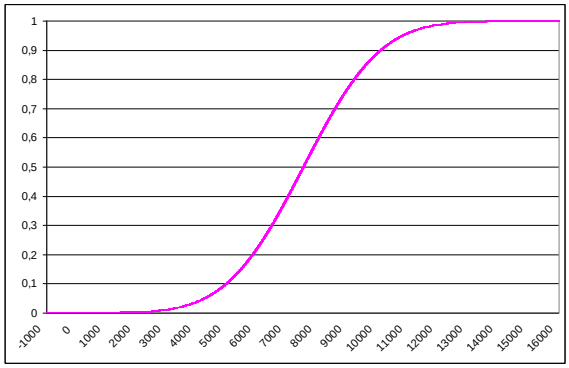

Figure 30: Normal F(x; 7500, 2000)

Log-normal distribution

$$F(x;\mu,\sigma) = \frac{1}{2} + \frac{1}{2} erf\left(\frac{\ln(x) - \mu}{\sigma\sqrt{2}}\right)$$
$$\mu = mean, \sigma = stdev, erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

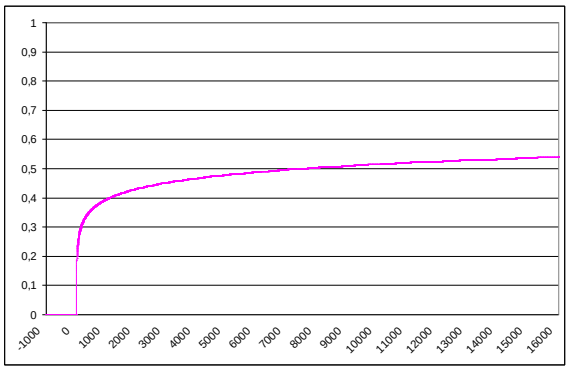

Figure 31: Log-normal F(x; ln(7500), ln(2000))

Logistic distribution

$$F(x;\mu,s) = \frac{1}{1 + e^{-(x-\mu)/s}}$$
$$\mu = mean, \; s = \sqrt{\frac{stdev^2}{\pi^2/3}} \qquad [44]$$

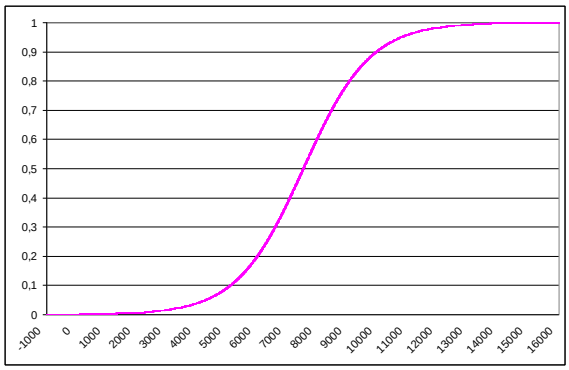

Figure 32: Logistic F(x; 7500, 1102.65…) such that stdev = 2000

Chi-square distribution

$$F(x;k) = \frac{\gamma(x/2, k/2)}{\gamma(\infty, k/2)}$$

$k = mean$, $\gamma =$ gamma function
(see Gamma distribution)

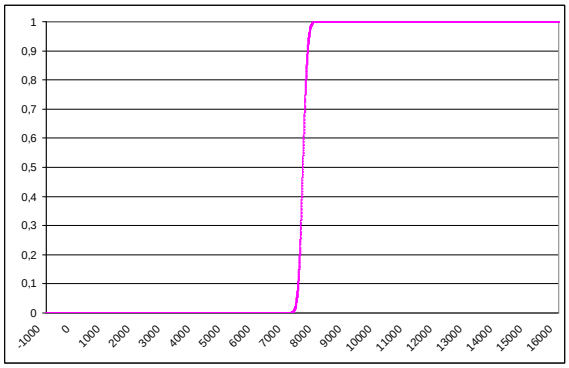

Figure 33: Chi-square F(x; 7500)

Uniform distribution

$$F(x;a,b) = \frac{x-a}{b-a}, \quad a \le x \le b$$

a = minimum, b = maximum

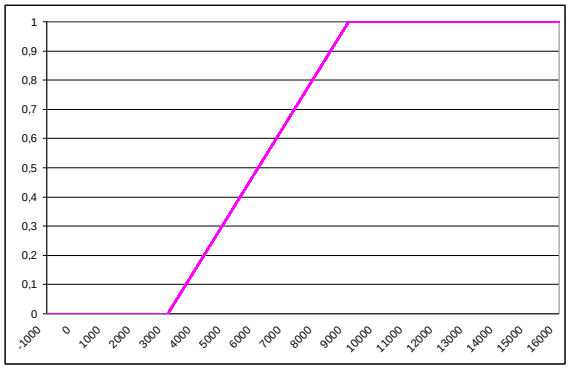

Figure 34: Uniform F(x; 3000, 9000)

Triangular distribution

$$F(x;a,b,c) = \begin{cases} \dfrac{(x-a)^2}{(b-a)(c-a)}, & a \le x \le c \\ 1 - \dfrac{(b-x)^2}{(b-a)(b-c)}, & c \le x \le b \end{cases}$$

a = minimum, b = maximum, c = mode

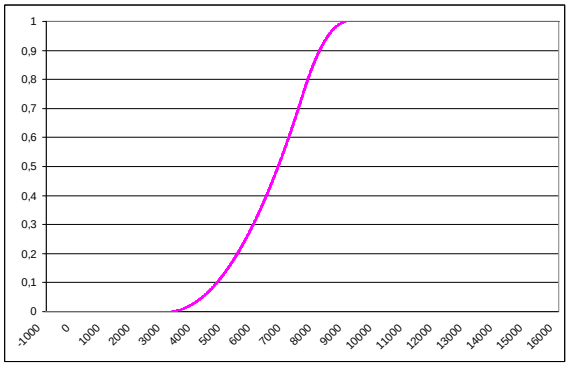

Figure 35: Triangular F(x; 3000, 9000, 7500)

U-quadratic distribution

$$F(x;a,b) = \frac{\alpha}{3}\left((x-\beta)^3 + (\beta-a)^3\right)$$

$$\alpha = \frac{12}{(b-a)^3}, \beta = \frac{a+b}{2}$$

a = minimum, b = maximum

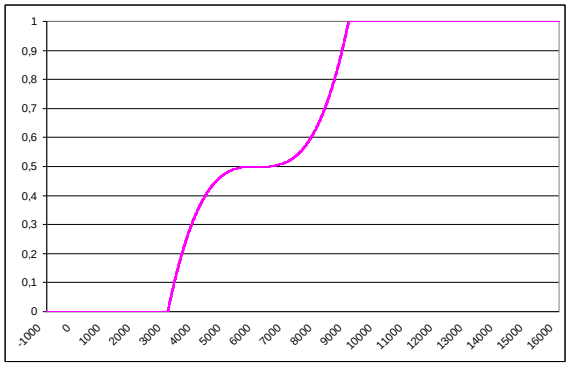

Figure 36: U-quadratic F(x; 3000, 9000)

Beta distribution

$$F(x;\alpha,\beta)=\frac{\mathrm{B}(x,\alpha,\beta)}{\mathrm{B}(1,\alpha,\beta)},\quad 0\le x\le 1$$

$$\mathrm{B}(x;a,b)=\int_0^x t^{a-1}(1-t)^{b-1}dt \text{ (beta function)}$$

$$\alpha \approx mean\left(\frac{mean\,(1-mean)}{stdev^2}-1\right)$$

$$\beta \approx (1-mean)\left(\frac{mean\,(1-mean)}{stdev^2}-1\right) \quad [44]$$

Gamma distribution

$$F(x;k,\theta)=\frac{\gamma(x/\theta,k)}{\gamma(\infty,k)}$$

$$\gamma(x;s)=\int_0^x t^{s-1}e^{-t}dt \text{ (gamma function)}$$

$$k\approx\left(\frac{mean}{stdev}\right)^2,\ \theta\approx\frac{stdev^2}{mean} \quad [44]$$

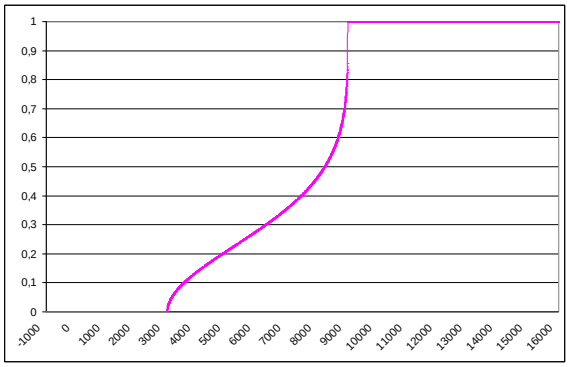

Figure 37: Beta F(x; 0.51…, 0.17…) with parameters estimated from mean = 7500, stdev = 2000, minimum = 3000, maximum = 9000

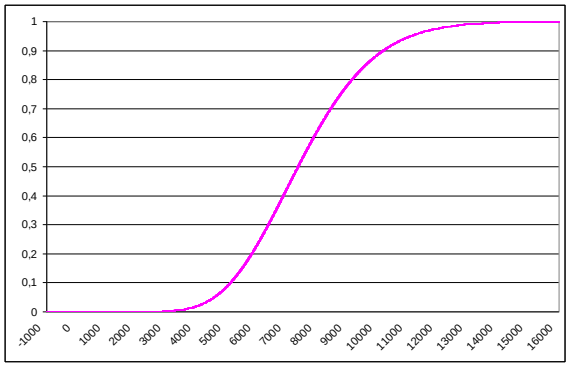

Figure 38: Gamma F(x; 14.06…, 533.33…) with parameters estimated from mean = 7500, stdev = 2000

Gumbel distribution

$$F(x;\mu,\beta)=\exp(-e^{-(x-\mu)/\beta})$$

$$\beta\approx\frac{stdev\sqrt{6}}{\pi},\ \mu\approx mean-\gamma\beta \quad [44]$$

$\gamma=$ Euler-Mascheroni constant (0.5772…)

Rayleigh distribution

$$F(x;\sigma)=1-\exp\left(\frac{-x^2}{2\sigma^2}\right)$$

$$\sigma=\frac{mean}{\sqrt{\pi/2}} \quad [44]$$

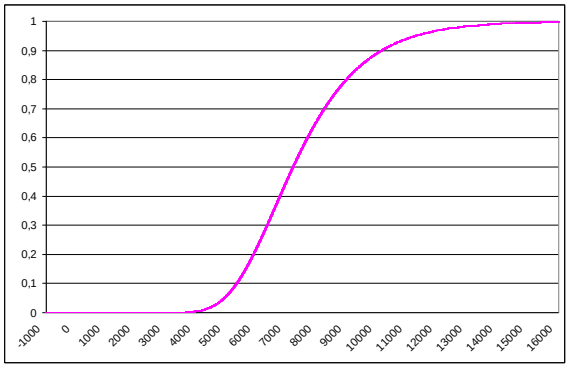

Figure 39: Gumbel F(x; 6599.89..., 1559.39...) with parameters estimated from mean = 7500, stdev = 2000

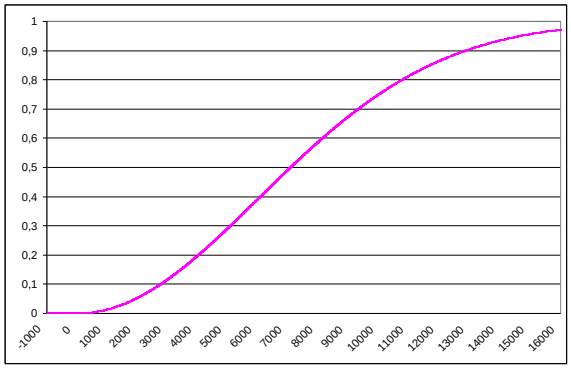

Figure 40: Rayleigh F(x; 5984.13...) with parameter estimated from mean = 7500