UNIVERSITY OF TWENTE.

DEUTSCHE BÖRSE GROUP

Software Performance Prediction in early Development Phases: A UML- and Simulation-based Approach

Felix Gehring, 2012

Master Thesis

Software Performance Prediction in early Development Phases: A UML- and Simulation-based Approach

by

Felix Gehring

supervised by

Dr. Christoph Bockisch

Dr. Chintan Amrit

Dr. Klaus Gebhardt (Deutsche Börse AG)

at the

University of Twente

Enschede, The Netherlands

June 24, 2012

Contents

1 Introduction

It should be every professional engineers endeavor to satisfy those needs that the customer poses on a product. He should also make sure to meet those requirements in a timely manner since the cost of change develops disproportionately over time. This is especially true for software engineering where projects fail more often than in any other engineering discipline.

Requirements are usually divided into functional and extra-functional requirements and it is widely recognized that the satisfaction of both is absolutely crucial for the success of a project. However, looking at the general software engineering methodologies it becomes obvious that the focus is often placed on meeting only the functional requirements.

One of those extra-functional properties is the performance of a computer system.

Over the last decade many authors claimed that performance is not considered in the traditional software engineering methodologies and that the existing methods that deal with the performance of software systems can rarely be integrated in the standard process with ease. Although there are methods available that support the analysis of the performance properties of a software system they often suffer from similar problems.

- Conducting the analysis can be very time consuming, especially if everything has to be done manually and no proper framework for doing so exists.
- If the method is dependent on a running system that can be measured and analyzed, it is not suitable for use in very early stages of the development process. This is also true for methods where the source code of an application is analyzed.
- Software developer may not be used to working with the models which makes the application uncomfortable for them. This can be especially true for proprietary and specialized models which are less commonly used in general.
- Engineers usually do not like to guess small details of the performance properties of the software they are designing, such as the mean response time of a single method. If a performance analysis is dependent on such detailed performance parameters and the method does not aid the developer in collecting the necessary data, engineers will dislike it.

We envision a methodology that enables system engineers to perform performance analyses at early stages of the development process, without creating too much overhead in the workload for the engineers, such that the analysis can be conducted even in small project teams with very limited resources.

Since with existing methodologies, performance analysis can be very resource consuming, both in understanding and applying the methods, it is often skipped completely. This puts software projects at great risk when performance issues are detected late in the development process and are expensive to fix.

Our approach is based on thesis paper by Marzolla (2004) in which he describes a thorough framework for software performance analysis and prediction. The author uses UML concept diagrams to generate a simulation from that represents the system under development (SuD) and makes it analyzable even before it is implemented. UML is a standardized modeling language that is utilized in many software projects.

Furthermore, we want to give an explicit guideline of how to base a performance analysis on the analysis of an existing system. Applications are seldom built as greenfield projects, thus base systems exist in many software systems. Such base systems can be great sources of information and system engineers may want to use this information in the performance analysis of a newly developed system rather than guessing performance parameters, as explained above.

Banks (1999) already stated that the analysis of a new system maybe based on a that of an existing system. We will use his approach for conducting a performance prediction study and describe this step of using a base model explicitly.

Eventually the adapted approach will be we applied in a case study at the Deutsche Börse AG, Frankfurt, Germany to evaluate the usefulness of the procedure and the techniques in a real project. The case study is conducted together with a software engineering team of the Knowledge Management department of the Deutsche Börse AG. They have been developing a web search engine called Xpider for several years now. The search engine is a distributed and multi-threaded system of considerable size. During the development of a new version (4.0) the performance engineering approach that is described in this paper shall be used.

1.1 Performance Engineering

According to Pooley (2000), performance analysis is the attempt to "understand and predict the time dependent behavior" of software and network systems. When performance analysis is executed as part of the software engineering process, in order to build a system that adheres to given performance constraints, this is usually referred to as performance engineering. In 1999, Smith coined the term "Software Performance Engineering" (SPE) for this concept of applying performance predictions early on in the development process.

The importance of conducting performance predictions has been emphasized by many authors. Smith & Woodside (1999) stated that a model-based analysis (cf. 1.2) should be undertaken prior to the implementation phase of a new software in order "to reduce the risk of performance failures" early in the software life cycle. This conforms with the generally accepted fact that "the cost of reworking errors in programs becomes higher the later they are reworked in the process, so every attempt should be made to find and fix errors as early in the process as possible" (Fagan, 1976). Also Balsamo et al. (2004) stress this point. They explain that "performance problems may be so severe that they can require considerable changes in the design, for example at the software architecture level."

Despite the apparent importance of performance engineering, several researchers claim that SPE is not as widely used in software projects as it should be. Smith & Woodside (1999) see mainly two reasons that may prevent the application of SPE in many projects. These are "the separation of performance engineering from development methods" and "the high skill level needed to apply them". In 2004, Balsamo et al. write in a survey-paper that several approaches to performance engineering have been successfully applied but that the level of integration of performance predictions into the standard software development methods is still low. Even later, almost a decade after Smith's and Woodside's publication, Abdullatif & Pooley (2008) state in their paper that these problems are still not solved, entirely. They say that performance modeling is usually only carried out by experts who have extensive experience in that field. Often, when these experts are not available in a software project, SPE is skipped completely, also due to missing budgets.

In fact, the authors say that performance engineering became even more difficult over the last years, therefore requiring even better-educated experts, mainly because of the advent of strongly distributed systems. Addullatif &

Pooley (2008) state that performance issues usually arise with the interaction of different components. The seriousness of this problem becomes apparent when the increased complexity that usually comes with distributed systems is considered, as stated by Briand, Labiche & Leduc in 2004. So distributed system are not only more prone to performance issues, it is also harder to fully understand such systems and thus detect problems early. This strong link between the understanding of a system and the susceptibility to errors has also been proven in (Matousek & Schneider, 1976) where it says that more than one third of all "engineering disasters" were due to the developers' insufficient knowledge of the target system. All this emphasizes the need for a systematic performance engineering effort throughout the development process of a software system.

1.2 Model-based Performance Prediction

There are basically two ways of getting reasonable performance predictions for a software system, or, as Arief and Speirs (2001) put it, "we don't know what kind of performance a particular design will deliver until we built a prototype or a model based on the real system". Hence, if possible a piece of software can be monitored during execution and the results can be extrapolated using the measurements. This was described by Bass, Clements & Kazman (1998) and Bosch & Molin (1999). As Balsamo et al. (2004) explain, this would only be an option in the later development and improvement phase of a software project where a measurable program implementation or at least a prototype is available. The same is true for approaches where software models are reverse-engineered from trace messages or other information sources of a running system (cf. Briand, Labiche & Leduc, 2004).
Arguably the better way of getting an idea about the performance of a software system as early as possible is relying on a model representation of the actual system "which reproduces the time dependent behavior of an unrealized system" (Pooley, 2000). Pooley states these four advantages of using models:

- An analysis can be performed even before a system is bought or even before it exists.
- In a real application it may not be possible to create the workload that a system should be able to manage in extreme situations.
- With monitoring the analysis is restricted to measures that are visible from outside an application whereas a model can provide arbitrary

measures.

- If a productive system is analyzed under extreme conditions it may break or even harm its environment. This is not possible when a system model is used.

The difficulties of model-based performance prediction match those of performance engineering in general. Pooley (2000) claims that the correct application of statistical concepts poses a major problem for many computer scientists. Such concepts are necessary "abstraction and approximation techniques" that are used in both, the analysis of observed data and the creation and analysis of performance models.
Another drawback of model-based analysis approaches lies within the used models. Depending on the utilized models a trade-off between the accuracy and the execution time must be made. Smith & Woodside (1999) warned that it may be tempting "to include excessive detail" but that a balance must be achieved. Furthermore it is not possible to represent every program in every model type. Users must be aware that certain problems may not be expressible in all models.

Often a system is expressed in both, a human-readable and a machine-readable format. The human-readable format of the system, also called the "concept model", is usually a visual representation of the software. Often such a model is already created prior to, or during the design and implementation phases of the software development. They help the engineers to plan the application and get a common understanding of how it should be built. If no such model was created in those early phases it might still be helpful to create it later in the process to document and analyze the design of an application. Often a standardized modeling language like UML is used for the concept model (cf. Section 2.1).

Since usually the model shall be solved or executed by a computer, it must also be available in a machine-readable format. In the context of performance engineering this also called the "operational model". While the concept model may already have a formal and machine-readable representation this is not required. In that case, a specialized operational model is needed. For these models there exist some alternatives which are suitable for different types of applications. Simulation, as the operational model that we chose in this approach is further elaborated on section 2.2.

1.3 Outline

This paper is structured in the following way. Chapter 2 provides information on the technology and methodology that was used as a foundation for our attempt at a solution. In chapter 3 we present the actual solution. Chapter 4 gives an evaluation in form of a case study in which the proposed solution is applied, before the work is summarized and a future outlook is given in chapter 5.

2 Technical Considerations

Some technical considerations were made in advance to the actual solution approach. The reflections especially concern the technological foundation of the solution. The decision making process that led to the final choice was first and foremost governed by a practical assessment of the alternatives and is briefly explained in the following sections.

We tried to make the approach as easy to use as possible for the project team at Deutsche Börse AG (DBAG), which is why usability and intuitiveness have always been the crucial factors.

2.1 UML Concept Models

UML diagrams are a prevailing way of depicting the structure and behavior of software systems in many development projects. The UML was standardized by the Object Management Group in 1997 and has gained widespread acceptance among software developers since then. One reason for that is the fact that UML is a language rather than a methodology which makes it usable as a part of many existing engineering methodologies. Additionally, it is totally independent of the programming language that it is used with which, again, makes it generally applicable (Bell, 2003). This has also been pointed out by Dobing & Parsons (2006) who stated that the language may not always be used as generally assumed but that it still facilitates communication among developers as well as between developers and business people.

UML models are usually grouped into three categories, namely behavioral, structural and interaction diagrams, where interaction diagrams are a specialization of behavioral diagrams.

Structure diagrams Structure diagrams are used to represent the architecture of a software system i.e. those physical elements that must

exist in the system. For example, class diagram, component diagram, and deployment diagram.

Behavioral diagrams Diagrams that show how the system functions and behaves. For example, activity diagram, and use case diagram.

Interaction diagrams Specialization of behavior diagrams with a focus on the communication that happens among the components of the system, such as classes and components. For example, communication diagrams, and sequence diagrams.

Performance engineering is concerned with the behavior of a software system over time. This is the reason why we will use the activity diagram, a behavior diagram, as the model for our approach. Of course, the functionality and the performance of a system always depends on the structure its structure, too, which is why other performance engineering approaches also utilize structure models such as deployment diagrams. For example, for the performance of a multi-component system it may be important to know which components read and write from one and the same hard disk drive or whether two servers are connected on the same network bus. For the sake of simplicity, we generally ignore the structure of the system as explained below in section 3.2.

2.1.1 Profile for Schedulability, Performance and Time

The UML profile component for performance modeling serves four related purposes. In the context of this paper support for "presenting performance results computed by modeling tools or found in testing" (OMG, 2002) is the most important one.
This UML component defines a set of concepts which are particularly useful in performance modeling:

Scenario A Scenario is a predefined sequence of steps that are executed and which are particularly interesting, for example to a customer. "Quality of Service" attributes are usually stated for complete scenarios.

Step A step is a part of a scenario which may require resource to fulfill its function. A step takes a finite amount of time to execute. In most cases a step can be seen as a small scenario in itself, as steps usually consists of a number of substeps, which may not be modeled explicitly.

Resource A resource is something that is needed by a step to execute, e.g. a processor or storage device. There is a distinction between processing resources and passive resources. Processing resources are those resources that are responsible for executing one or more steps. Passive resources may be shared among and access by multiple operations and are protected by some form of access mechanism.

Workload The demand for one particular scenario. The expected or required response time for that scenario may be specified, too. Only the topmost scenario may have a defined workload (no steps/subscenarios). A distinction is made between open and closed workloads. Open workloads are modeled as a "stream of requests that arrive at a given rate", whereas closed workloads represent "a fixed number of active or potential users or jobs that cycle between executing the scenario and spending an external delay period [...] outside the system."

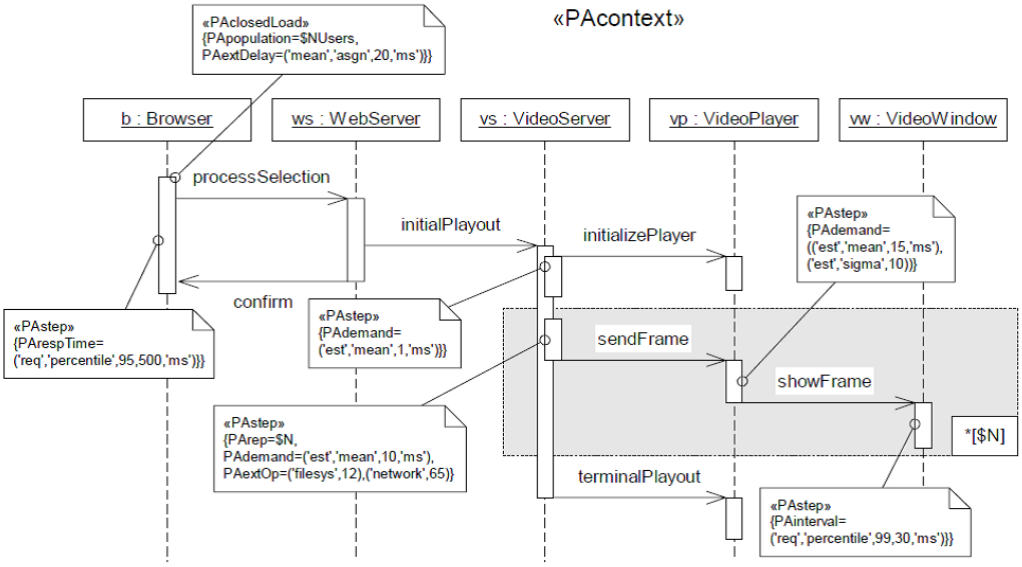


Figure 1: Performance Annotated Sequence Diagram (OMG, 2002)

All concepts may be augmented with certain attributes, such as the response time for steps or the scheduling policy for resources. These attributes should allow for a performance analysis of the scenario or set of scenarios. These concepts are represented in additional features for the standard UML diagrams deployment diagram, sequence diagram and activity diagram. Below a so-called performance annotated sequence diagram is depicted. Notice

that the performance annotations are prefixed with "PA", e.g. "PAclosedLoad" and "PAstep".

2.2 Simulation

Several operational models, also called performance models, are available for representing the system under development in a machine readable format. These models include, Extended or Layered Queuing Networks, Process Algebra, Petri Nets, Stochastic Processes and Simulation. According to a survey (Balsamo et al., 2004) Queuing Networks are the most frequently used performance models due to their high abstraction level which makes it possible to represent software system on a characteristics level rather than implementation level as a set of communication patterns and hierarchical structures.

Among the scientific work that we are basing this paper on, simulation plays an essential role. Many researchers have presented practical applications where simulation was used a the operational model. Apart from that, simulations have some properties that we consider advantageous in our situation, but also a few risks that we are trying to mitigate.

Constructing a simulation is comparable to building a software application with the comfort that complex tasks within the system can easily be hidden in a simple sleep command that emulates the processing time, without sacrificing accuracy regarding the time-dependent behavior of the system's representation. This is also the reason why software developers, who supposedly are the main users of the approach, may like simulation more than highly mathematical models.
This point is related to the fact that simulation is also the most powerful performance model. Constructing a simulation is not only like programming the system, it is also possible to simulate every behavior that can be programmed into an application in terms of its time-dependent behavior.

However, this power also comes at a price. Since a simulation is executed by a simulation framework rather than being solved mathematically, this execution may take a considerable amount of time. The sheer duration may render very complex simulations impracticable, although this problem becomes negligible with the constantly increasing speed of modern processors. Additionally, the potentially great level of detail also bears other risks. Developers may be tempted to put excessive detail into those parts of the model that represent a subsystem that they know especially well which may be bad

for the analysis of the simulation's results. In particular, if one part of the system is modeled in considerably more detail than others, the result may be biased towards this part. Furthermore this may give the analyst a false sense of precision since a very detailed model is not necessarily very precise.

2.2.1 Different Design Approaches

Two of the most commonly used simulation types are continuous and discrete-event simulations (Helsgaun, 2001). They differ in how the states of the simulated system changes over time. Discrete-event simulations are particularly suitable where the simulated system is subject to discrete state changes triggered by distinct events. An example for this is the one-teller bank taken from Banks (1999). Customers arrive at the bank at a specific rate and are served one after the other. Whenever the service time is longer than the time between two customer arrivals a customer has to wait. In this example, the arrival of a new customer, the beginning and the end of a service are events on which the state of the system as a whole changes.
Continuous simulations can be used for systems where the state changes continuously over time. An example for that is the simulation of a population of predator and prey. The number of predators rises with a certain delay after the number of preys rises. But when there are more predators, the number of prey soon diminishes. Since there is less food available now the population of predators also declines, which makes the process start over again, since this automatically means a rise in the population of the prey. Note that we do not consider individual births and deaths as events in the system but the development of the population in its entirety.

Naturally, binary computer systems are discrete-state systems, where components are either on or off. Therefore they can always be modeled with a discrete-event simulation. However, one could think of processes which are better modeled as continuous processes, such as the transfer of files over the network. The size of the transfered files on the downloading machine increases continuously over time. For our purposes, however, it makes sense to abstract from these apparently continuous processes and only consider the beginning and the end of the processes as distinct, discrete, and stage changing events. That is why we will use discrete-event simulation exclusively in our approach.

Within the theory of discrete-event simulation there exist three basic approaches as to how the simulation represents the real system and how it is implemented (Helsgaun, 2000). Using the event-based approach, a model

consists of a number of explicitly defined events. These events happen instantaneously and do not require time. Each event is schedules the occurrence of successive events. In the one-teller bank example the time when a customers is served is an event which does not require time itself but schedules the next event, namely the end of the service to occur some time later.
The activity-based approach defines the model as a collection of so-called activities. Activities must define actions that are executed at the start of an activity and actions that are executed after its execution. Whether an activity is started depends on the starting condition which is defined for every instance just like the duration which represent the time between the start and the end of an activity. Due to the constant examination of the starting conditions of non-active activities by the simulation framework, this approach can be quite demanding on the resources.
In the third, the process-based approach, every real-world entity is represented in a so-called process. Processes hold for a certain amount of time, representing processing time, can activate other processes and can suspend their activity until it is activated again. Once terminated a processes cannot be started again. This approach is usually considered the most intuitive approach, because of its direct mapping from real-world entities to simulated processes.

In the following work we will use only the process-based approach due to its immediacy and intuitive usage. Extensive programming libraries and existing literature for the process-based approach also allows for easy implementation in the Java programming language.

2.2.2 Implementation of a Simulation

SIMULA (SIMUlation LAnguage) was an early attempt to facilitate the design of discrete-event simulation with a programming language based on ALGOL 60. It was presented in 1966 by Dahl and Nygaard. The concepts of SIMULA are widely used in the implementation of simulations. Dahl and Nygaard introduced the notion of a process as the representation of the life cycle of an entity in the real world. It serves as a data carrier and executes actions.

A process can be in one of four states: It can be either active, suspended, passive or terminated. Only one process can be active at an instant of time. Processes that are scheduled for activation at a later point in time are suspended. Passive processes must be activated or scheduled for activation by other processes before they are executed again. Terminated processes cannot be activated again and mark the death of the represented entity. The active

process and all suspended processes are contained in the so-called sequencing set, also called the future event list, in non-decreasing order by their scheduled execution times. The active process is always the head of the sequence.

The ideas of SIMULA have also been implemented in Java, by Helsgaun (2000). The author developed a library called javaSimulation which can be used for discrete-event simulation in Java and aims at providing the complete functionality of SIMULA. Java is by far the most commonly utilized programming language by the project team in the knowledge management unit of the DBAG.

For the process-based approach to simulation all the functionality is implemented in the class 'Process'. The method signatures of this class match the process states described above very well. By calling hold() on a process it is suspended and enqueued into the sequencing set o be activated again after the given amount of time, whereas a call to passivate() makes the process wait for the explicit activation by another process (activate(p)).

Whenever a process is activated its actions() method is executed, Once this method finishes, i.e. there are no more actions to be executed by this process it is considered dead and cannot be activated again.

The basic concept of SIMULA requires that a process can be suspended during execution and continued later. Such processes are generally referred to as coroutines (Conway, 1963). Since there is no comparable concept available in Java, i.e. Java methods cannot be suspended during runtime, this is emulated in the javaSimulation framework by using threads. When a Java method that implements a simulated process encounters a hold() or passivate() instruction, the underlying thread is suspended, therefore holding the methods execution. Consequently, every process runs in its own thread.

Using Java threads to emulate coroutine behavior, however, comes at a price. The excessive creation and deletion of Java objects slows down the execution of an application, so the creation of vast amounts of threads must be avoided. javaSimulation has an inbuilt mechanism that recycles thread objects for the simulation of multiple processes when one process dies. Still, as Helsgaun points out the efficiency of thread usage is also dependent on how the simulation and its processes are implemented. So, especially in projects of considerable size it may take skilled programmers who know the javaSimulation package well to implement swift simulations.

We will base our methods on the javaSimulation package because it provides the well-established concepts of the SIMULA language and the complex functionality that the Java implementation of those concepts requires.

2.3 Conducting a Prediction Study

The general approach of executing a prediction study is usually consistent with a generic engineering design evaluation approach. This was described by Smith & Woodside (1999) who presented a five-step approach to accomplish the prediction:

1. Determine the requirements and how the system is supposed to work and be used.
2. Find an underlying structure of the system and develop a model that is able to represent the performance characteristics of the system.
3. Collect performance parameters, i.e. resource requirements of the processes.
4. Execute the performance model and check in how far the requirements are or will be met.
5. Analyze the outcome and see how those parts of the system can be improved that impede the overall performance.

Within this approach a validation of the system under analysis, that is the check whether the system meets the given performance requirements is performed in step 4. In case this validation fails, a loop is entered in step 5 and the procedure is started from the beginning, giving the architects a chance to improve the system such that it finally passes the test.

These five general steps can be found in other approaches as well. In what follows we will briefly introduce four models which, although not explicitly based on the model by Smith & Woodside, can easily be mapped into the five steps. Essentially the authors provide more detailed and focused approaches, thus introducing additional steps to the performance prediction but still adhere to the basic stages. Figure 2 illustrates this fact. The models we examined were Banks et al. (2001), Shannon (1998), Law (2003), and Williams & Smith (2002).

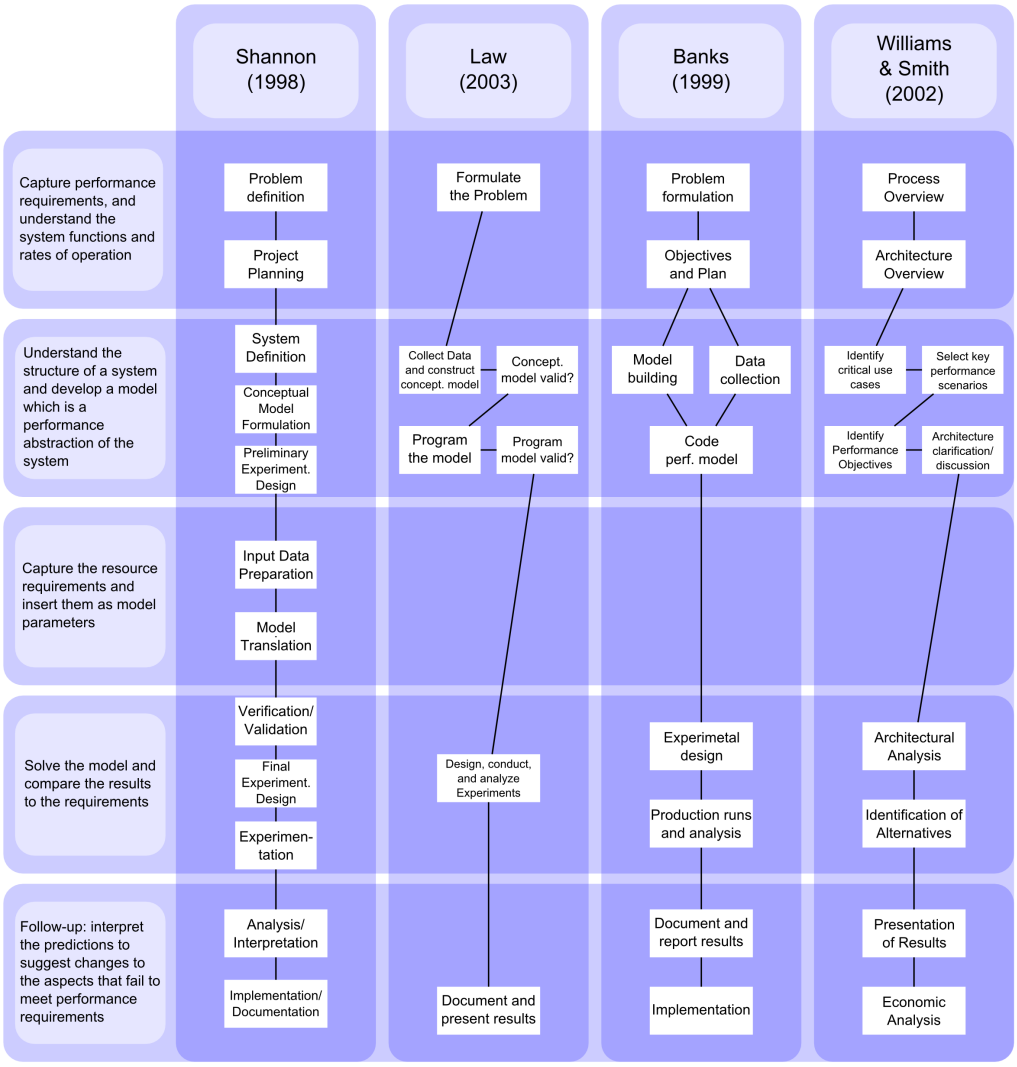


Figure 2: Comparison of different Prediction Study Approaches, based on the five Stages given by Smith & Woodside (1999)

Strikingly, only one of the four approaches seems to prescribe steps in the third phase "Capture the resource requirements and insert them as model parameters". Only Shannon describes steps that can be mapped onto this phase, immediately. Looking closer at the model it becomes clear that this impression is deceptive. As Smith & Woodside already point out themselves, these steps are often inseparably included in the earlier stages of the analysis. Law covers the data collection in the second step, at the same time as the creation of the first system model. Banks also depicts the collection

of performance data as being executed concurrently with the construction of a model. Only the PASA^{SM} framework by Williams & Smith shows no comparable step. The reason behind this is that the authors describe a more abstract approach here which is not primarily concerned with distinct performance parameters. The goal of that framework is the assessment of a high-level software architecture where the basis for the assessment is usually of qualitative rather than quantitative nature. Williams & Smith state that only critical portions of a system must be analyzed quantitatively, in which case a suitable quantitative approach shall be embedded in the process.

Furthermore, it becomes obvious that the authors put a special focus on the second stage of the process, since this is the stage where, in all approaches the most steps are defined. This matches the generally agreed upon difficulty of this project phase. Abdullatif & Pooley (2009) even go so far as to say that the modeling process can be considered an art.

For our approach we selected the framework by Banks for two reasons. (1) The work by Banks is extremely well recognized in academia. It has been cited around 3000 times and therefore outperforms the other authors by far. (2) Marzolla provides great insight into how the framework can be used in an environment where UML is the predominant modeling language.
In the following the different steps of Bank's approach are briefly described.

Problem Formulation The problem that shall be solved with the help of the simulation must be clearly stated such that both the analysts and the clients understand it and agree with it. If the problem statement turns out to be unsuitable in the course of the project it may have to be reformulated.

Objectives and Project Plan Based on the problem statement, the objectives of the study must be stated in this step. Additionally, a plan must be prepared that explains how the goals shall be achieved. That plan has to include the different scenarios that will be investigated and which resources will be needed to do that. Needed resources should be stated in terms of time, money, personnel, hardware and software required to successfully conduct the study. Other information such as billing procedures may be included if applicable.

Model Conceptualization In this step the system under development is represented in a model. Banks suggests that the model is developed

with increasing complexity over time and be oriented towards the specific problem at hand. This way an overly complex and thus expensive construction of a system model shall be prevented. Furthermore it is recommended to develop the model together with the client who should have the clearest picture of the actual system.

Data Collection The data collection step can also begin after the project plan has been agreed upon, it can therefore be executed in parallel with the model building. During this step the data to parameterize the model and to validated the model later must be collected. Banks points out that it may not be enough to collect the information but that very often the raw data must processed first to be useful. What kind of data should be collected is explained by Smith & Williams (2002). Depending on the application the following types of data may be collected: (1) Workload data that describes how frequently a certain scenario is requested, (2) data characteristics which describe the amount of data being send through the system and the corresponding frequencies, (3) execution characteristics that provide information about the duration of individual processes and the different execution paths through the system as well as their probability of actually being executed, and (4) the so-called computer system usage information which is data like the complete scenario response time or the throughput, which is generally useful for validation of the performance model.

Model Translation This is where the conceptual model is translated into a computer-readable model that can be executed. Banks names the verification process explicitly here, where the executable model is checked for errors that could hinder the simulation software in carrying out the simulation. Additionally a validation should take place in this step which means that the analyst should make sure that the model really represents the desired characteristics of the system. Here it is suggested that a base system may be used to validate the simulation against. We will focus on this aspect in the following Section 3.1.

Experimental Design The context of the execution is set here, i.e. the length of the simulation, its initialization and the number of repetitions.

Production Runs and Analysis This is where the scenarios that were determined previously are finally executed and analyzed. Based on the result the system analyst gets an idea about the performance of the individual scenarios.

Documentation and Reporting For the analysis to be useful for other analysts and developers it has to be well documented. Also, if the models or the data changes later on in the performance analysis, a good documentation can come in handy. Moreover, the results have to be presented in a way that the client understands the effect those results have on the development process and where the spent resources went in the study.

Implementation Although not directly part of the performance analysis Banks stresses the point that the implementation of the actual system is an important part of the process. After all, the performance engineering process is supposed to be just a small step towards the successful production of a software that meets the requirements.

In the solution approach we will use Banks' framework and extend it to suit our needs.

3 Solution Approach

Our solution approach consists of five parts. First, we introduce an adapted approach to conducting a performance analysis of a software system with the help of a simulation. This adapted approach makes the analysis of a base system as the foundation for the performance analysis of a new system explicit in that it prescribes steps to be executed by the system analyst. Secondly, we describe the elements of the concept model that can be used to describe the behavior of a software system in a human-readable, consistent and translatable way. Thirdly, we establish a set of building blocks that can be used to implement a software simulation with. These blocks are actually implemented Java base classes resting upon the javaSimulation framework which shall aid programmers in developing a simulation quickly. Fourthly, we provide a mapping of concept model elements to simulation elements. The mapping consists of a description of how to implement simulation processes with the given Java classes and make them an executable representation of the behavioral UML model. Lastly, we give hints towards how analyzable results can be obtained from the executing simulation.

Together these parts should give system developers the methodology and the tool kit to conduct a simple, yet reliable performance analysis at early stages of their development cycles.

3.1 The adapted Performance Analysis Procedure

The goal of this study is to allow system analysts to validate a system's performance model in a reliable and intuitive manner. Our solution to this problem takes into consideration that a base system or an existing, earlier version of the system exists, and that this base system may be used as a means of validating a model, simply by comparing the output of the model execution to the measurement data of the real system. Our idea is to incorporate this validation into the performance engineering procedure. After that, the model of the new system for which we want to make predictions can in turn be based on the, then validated base system model. Therefore, the new model is not validated itself but is based on/derived from a previously validated model.

To achieve this we extend the performance engineering framework by Banks. We add steps that explicitly require a system analyst to model the base system, validate it and derive the new system's model from it.

Figure 3 depicts the new, adapted approach. As one can see, the steps "Model building", "Data collection" and "Model translation", including the verification and validation are executed twice, once for the existing and once for the new system. The most important changes in the approach in comparison with the original one are within steps with the wide borders.

Validate base system's model The base system's model can be validated by comparing the simulation results to the measured results from the real software. The measurements of the real system should have taken place in the data collection step. If these results do not match, the model and the parameters have to be double-checked and improved where necessary.

Analyze base system's model This step is further elaborated on in section 3.1.1. It can be considered the most important but also the most difficult step in the adapted approach. The goal of this step is to find parts in the performance model of the base system that can be used to construct a reasonable model of the SuD. Reusable parts may include portions of the system models and corresponding performance parameters. If generally a portion of a model can be reused but the implementation and thus the performance parameters will change in the new system, educated guesses have to be made taking these changes into consideration.

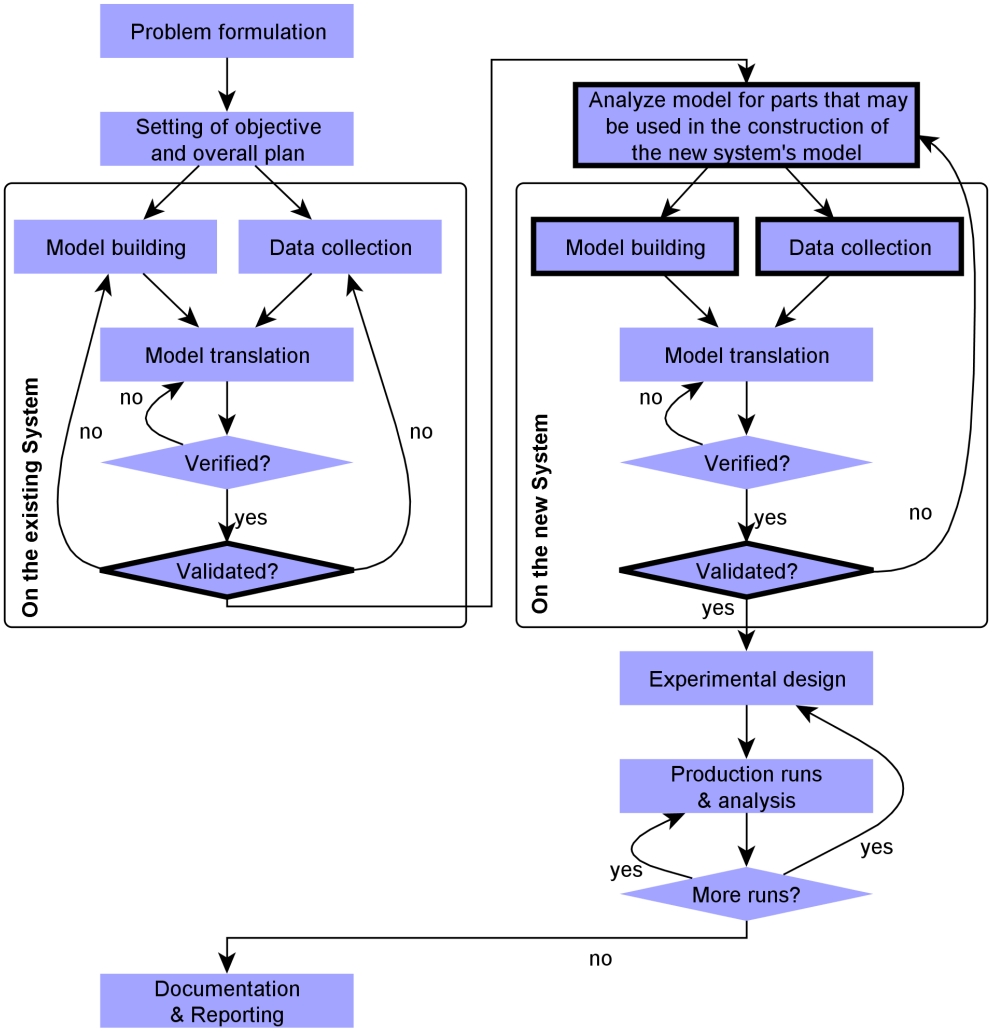


Figure 3: Adapted Simulation Study Approach

In case no reusable parts can be identified this approach does not offer any benefits.

Build SuD model The construction of the model of the SuD must be partly based on the base system's model as the previous step should have revealed. The bigger the portion of the model that is reused the higher the likely quality of the new model. Wherever information is missing, i.e. the base model cannot be reused, additional information must be collected.

Collect data for SuD model In this step, basically the same principles hold true as in the model building step: Wherever possible data from the base model shall be used. The complexity in this step stems from the fact that even minor changes in the implementation of the software may result in considerable changes in the execution behavior of the program. Again, educated guesses must be made in these cases as to how the data must be adapted.

Validate SuD's model The validity of this model can hardly be ensured. As pointed out by Marzolla (2004) neither formal proof nor proof by using test cases is applicable. Therefore, in this approach we rely on the soundness on the base model and only check for plausibility in this step. If the outcome of this check seems odd, we start over with the analysis of the base model as explained above.

3.1.1 Analysis of the Base Model

This step in the approach is the most important one. The goal here is to find parts of the validated base model that may be used to construct a valid model of the new system. Reusable parts may include portions of the model and key figures, for instance about the scenario that is examined.

This may, however, not be an easy thing to do. Depending on the granularity of the base model reusable parts may not be easily identified. It is important to keep the analysis of the base model on a high level, such that potentially confusing details do not complicate the analysis and make it difficult to identify similar components. Extremely detailed models are often hard to apply to different systems since they easily break on small changes. Since, according to Starfield (2011) a greater level of detail rarely improves the accuracy of a systems representation, and more detail can always be added later, the analysis of whether a part of a model is transferable or not, can be carried out on a rather abstract level.

We introduce the following three substeps to the analysis of the base model. These substeps should guide engineers and help them not to get lost in the details:

1. Identification of reusable portions of the model on an abstract component/service level.

2. Adaptation of these submodels in more detail, for instance adjusting individual response times.

3. Analysis of the execution order and frequency of the identified components/services.

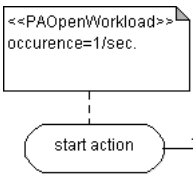
3.2 Elements of the Design Model

As explained above the design model is a human readable representation of the performance behavior of a software system. In this work it is considered to be a collection of information about the system which is representable in a performance annotated UML diagram.

The model must contain enough information to construct a performance model from but should still be easy to construct. That is the reason why all the information we use can be shown in an annotated activity diagram which turned out to be a very intuitive way of representing the execution of a program. However, activity diagrams do not provide the means to depict information about resources in the system. This can only be done in deployment diagrams.

We decided to ignore resources and therefore data from deployment diagrams for the following reasons. Resources like processors and their behavior are difficult to understand for non-experts. For instance, the scheduling policy or the context switching time of a processing resource may seem overly complicated to some engineers. Secondly, the demand of a process on the processing resource is almost impossible to determine in isolation, since there are usually many processes running on the same host in parallel. So even if we are able to measure the duration of process executions in a base system, these values hardly ever represent the individual demand of that process but usually incorporate some resource-sharing already. In turn this also means that we might get an accurate representation of the SuD without explicitly modeling resources because the effect of resource-sharing can be part of the measured or estimated response times, as it is.

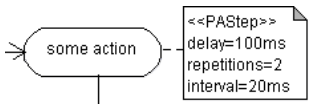
Listed below is the information that we need to collect in a design model to construct a simulation from.



Workload The workload that triggers the execution of the scenario which is represented in the activity diagram. We only consider so-called open workloads here, again for the sake of simplicity. Open workload means that the scenario is triggered a potentially infinite number of times at a specified rate.

occurrence The occurrence pattern according to which the first step of a scenario is triggered. This is usually represented as a random amount of time between subsequent requests.

firstAction A reference to the first action in the scenario which is executed on triggering of the scenario.



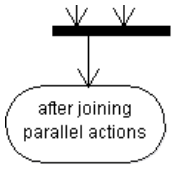
SimpleAction A processing action that requires time to execute. Subsequent calls are processed one after the other.

responseTime The time it takes to process a call. Usually a random number according a predefined distribution function.

repetitions The number of times this action is repeated per call.

interval The time between the execution of two repetitions. If repetitions is zero, this is ignored.

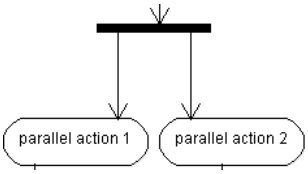
successor A reference to the following action which is called after all repetitions of this step have been executed.



JoinAction Joining of threads that are previously running in parallel. Waits until all preceding actions have finished.

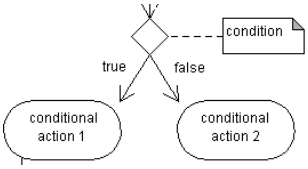
noOfPredecessors The number of preceding threads that must be joined.

successor A reference to the following action which is called after this step has been executed.



ForkAction Start of the parallel execution of multiple threads.

successors A set of successive actions that mark the start of the threads which will to be executed in parallel.



JunctionAction Conditional junction that gives control to only one of its successors depending on some predefined condition.

condition/successor pairs A set of pairs of conditions and actions which are executed if the corresponding condition is true.

3.3 Elements of the Simulation

Our implementation of the simulation is based on the javaSimulation framework. It was developed in 2000 by Helsgaun and emulates the functionality of the programming language SIMULA as explained in section 2.2.2.

The logic of the simulation is contained in one class. It's interface is as follows:

Listing 1: Simulation Main Class Interface

```
public class Simulation extends Process
{
    Random r = new Random();

    QueueProcess start_action = new QueueProcess();
    QueueProcess some_action = new QueueProcess();
    ....
    ....
    QueueProcess end_action = new QueueProcess();

    @Override
    protected void actions();

    private void report();

    public static void main(String[] args);
}
```

This class is an executable Java class with a main-function. It is responsible for initializing and running the simulation, stopping it and giving out the data for the analysis. This functionality is covered in the three methods main() where the simulation is started, actions() where the context of the simulation is set and the first action is called, and report() where the analysis data is printed after the simulation has finished.

Furthermore, a global variable of type Random is defined which can be used by every actions of the simulation. The advantage of having one global random number generator is that a seed may be given to this one Random object, to make the whole simulation repeatable with the same result.

The actual logic of the simulation is contained in the QueueProcess variables. Every action that is defined in the design model has its own QueueProcess representation. This class is further elaborated on in the following section.

3.3.1 Class QueueProcess

The class QueueProcess is based on the class Process which is part of the framework. It inherits all its behavior and must implement the actions() method which is called by the framework upon activation.

QueueProcess extends Process's functionality by the support for queuing. It stores the size of the queue and allows other processes to add a number of elements to it. With this approach, only a minimal representation of the queue, namely its size is stored. This could be changed by adding a real queue of objects. This is easily implemented by using javaSimulation's Head class, but may also be realized using other data structures to store the queue. The advantage would be that the queued objects could also contain some logic which may aid the analysis of the simulation results. However, for this to be useful the QueueProcess class would need to be manually adapted to handle the queued objects, for example notify them when they are processed. Also, by only storing the number of elements in the queue we are independent of any scheduling policies.

Every QueueProcess 'lives' until the end of the simulation. This means that it can be activated over and over again. 'Dead' processes can not be activated again. This is ensured by the infinite loop that surrounds all instructions within the actions() method. However, the execution of this process is put on hold when there is nothing to be done. This is achieved by the call to passivate() which blocks the execution until another process explicitly activates it again.

Whenever the queue is not empty and the process is activated it works until the queue size becomes zero. For each queued object, loopAction() is called. This is an abstract method which has to be implemented differently for every process.

The simulated execution time of loopAction is kept track of automatically. The total time that the process was running is stored in the variable active. After the simulation has finished this value can be easily used for a quick analysis of the result. It is made available through the public methods getActiveTime() which returns the absolute time, and getActivePercentage() which returns the active time as percentage of the total time that the simulation was running.

3.3.2 Class ControlProcess

For processes that do not required queuing, because they do not really process anything but only control the execution of other processes we introduce the class ControlProcess. It has a similar interface to QueueProcess but does not implement any queuing functionality. Nevertheless, this class also has a function loopAction() which is encapsulated in an infinite loop such that the process never 'dies'. Just like in QueueProcess the call to passivate() blocks the execution of this particular process until it is explicitly activate again by another process.

Control processes are used to model execution paths in the application, for instance junctions between alternative paths implement some logic but do not consume processor time, otherwise a ControlProcess may be used in conjunction with a QueueProcess.

3.3.3 Process Instances

A QueueProcess or ControlProcess instance should be created as a variable in the Simulation class as explained in the beginning of section 3.3. This procedure eases the communication among different processes despite the atypical design of the object-oriented program.

The logic of every process is implemented in the loopAction() method which must be overridden. QueueProcesses and ControlProcesses may be mixed with standard Processes if desired since they are just specializations of the Process class.

3.3.4 Multiplicity and Replication of Servers

An important technique to improve the performance of a software system is to install multiple server instances with the same functionality across which the workload is distributed (Franks et al., 2012). By multiplying a server and distributing the load the throughput may be improved, and the response time decreased. Additionally an overload of individual instances can be avoided in case of unexpected workload peaks.

Generally, there exist two ways of adding copies of servers, namely replication and multiplication. They differ in the way that requests are queued and forwarded within the system. When replicating a server, each server instance has its own request queue that it needs to process. When the servers are multiplied, in a so-called multi-server system, there is only one queue that

is used by every instance. The difficulty of replication lies in the fact that the different queues of the servers must be filled according to some previously specified convention. A preceding process may enqueue an object into every server's queue, which might make sense in a system where failure safety has a top priority. If an object is enqueued only into one server's queue there still has to be some kind of agreement which one of the queues is to be filled. For this reason, we will only consider multi-server systems and disregard replication of servers.

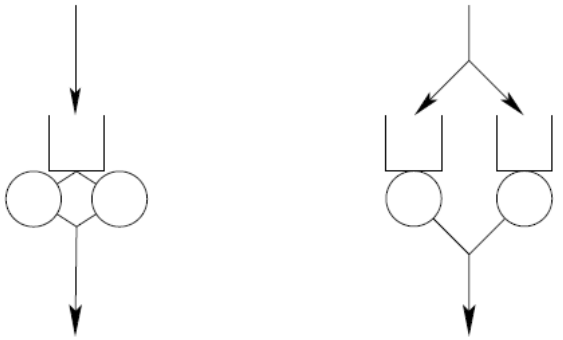


Figure 4: Multi-server vs. Replication (taken from Franks et al., 2012)

The multiplicity of servers can be represented using the construct from Listing 2. We call server instances 'worker'.
A multi-server component is modeled using a QueueProcess and an arbitrary number of incorporated ControlProcesses, the workers, that do not have their own queues. Furthermore two lists of workers are implemented, one containing all available workers and one holding the busy workers. A constant (final) integer value represents the maximum number of allowed workers in the system.

The workers are managed from within the QueueProcess which is implemented as an anonymous class like most other processes. This is the only process that is called by external processes.

Basically the actions of the QueueProcess can be described as follows: If a worker is available (not busy) it is activated. If there is no available worker, one of the two actions is chosen. If allowed by the defined maximum number of workers, a new worker is created, if not it waits until one of the busy workers finishes its actions and becomes available.

The workers are responsible for marking themselves busy upon the start of their processing time and available when they are done.

Listing 2: Multiplicity of Servers

```
QueueProcess worker_do_something = new QueueProcess()
{
    final int NOOFWORKERS = multiplicity_of_servers;
    Head available_set = new Head();
    Head busy_set = new Head();

    // local class extending ControlProcess and
    // representing the server instances (called 'workers')
    class WorkerProcess extends ControlProcess
    {
        @Override
        protected void loopAction()
        {
            // mark self as busy
            into(busy_set);

            hold(processing_time);
            successor.activate();

            // mark self as available
            into(available_set);
        }
    }

    int availableCardinal = 0;
    int busyCardinal = 0;

    @Override
    protected void loopAction()
    {
        // wait until a worker is available
        while (available_set.cardinal() <= 0)
        {
            availableCardinal = available_set.cardinal();
            busyCardinal = busy_set.cardinal();

            if ((availableCardinal + busyCardinal) < NOOFWORKERS)
                // create new worker if allowed
                new WorkerProcess().into(available_set);
            else
                // wait for some time (must not be zero)
                hold(10);
        }
```

```
        // activate the available worker
        ((ControlProcess)available_set.first()).activate();
    }
};
```

The multiplicity of servers is a concept that is only available in the simulation but not in the UML concept model because activity diagrams cannot represent this kind of information. We suggest simply adding a note to the corresponding activity in the diagram containing the required information.

3.3.5 Statistical Concepts available

Basic knowledge of statistical concepts is indispensable for the execution of simulation studies. This holds true for the construction of a simulation as well as for the analysis of the results.

As we said before, the execution times of different processes are usually approximated and emulated by means of random numbers drawn from certain probability distributions. Also the frequency of different execution paths being executed, usually follows a statistical model. Since we are assuming that a base system exists which is used to collect some real performance data we want to find the most suitable distribution to represent the real values.

It may seem tempting to simply abstract from the real values by using only the mean value rather than finding a suitable distribution function. However, this does not hold as a valid abstraction in most cases as it does not account for the uncertainties that usually exist in such systems (Shannon, 1998). In fact, taking constant values as input parameters would make many problems quite trivial to solve, even by hand. It is one of the strengths of simulations systems that they are able to calculate many different scenarios with different (random) input data in a very short time.

The javaSimulation framework offers some random drawing functions which return pseudo-random values according to an underlying probability distribution. In the following, we will present the most important probability distributions that are offered and explain how they can be used to represent measured or estimated data. In square brackets the corresponding method in javaSimulation is given.

Fair or biased coin toss [draw()] Returns a boolean value, true or false.

The probability for a "true"-event can be given as a given as a parameter where 0.5 describes a fair coin.

Uniformly distributed integer [**randInt()**] Returns a random integer value between the upper and lower bound which are given to the method as parameters. Every value between those bounds is equally likely to be drawn.

Uniformly distributed doubles [**uniform()**] Like randInt() this method takes a lower and an upper bound as parameters and returns a value in between those bounds. However this method works with double-precision floating-point numbers rather than with integers.

Normal distribution [**normal()**] Returns values according to a normal, Gaussian distribution. The method takes the mean value as well as the standard deviation as parameters. Mean and standard deviation are easily calculated from a sample of measurement data. In nature the normal distribution is frequently encountered, e.g. the body height of human beings is assumed to be normally distributed.

(Negative) exponential distribution [**negexp()**] This method returns double values according to a negative exponential distribution function. Most often this function is used to represent the interarrival times of events in a Poisson process. In a Poisson process events occur continuously at a constant rate and independently of one another. The only parameter that this functions takes is the inverse average interarrival time which is why this method is very easy to use given some recorded or estimated data. Accordingly it is probably also the most frequently used distribution function.

Poisson distribution [**poisson()**] The Poisson distribution is strongly connected to the exponential distribution function. While the exponential distribution function is used to represent the time between successive events the poisson() method returns the number of events per unit time and takes the average number of events per unit time as its only parameter (cf. Bruck, 2006). Again this is only a valid model for Poisson processes, as described above. The number of requests for a specific documents on a web server has been found out to be Poisson distributed (Arlitt & Williamson, 1997).

As we described above, the negative exponential as well as the Poisson distribution expect the events to satisfy the Poisson characteristics:

- Events happen independently of one another.
- The probability of two events happening almost simultaneously is negligible.
- The probability of the occurrence of an event is only dependent on the length of the interval, not on the current time.

Additionally it is often said to be particularly useful with small numbers (Waring States Projects, 2007).

A well-known example of a Poisson distribution is the number of deaths per year due to horse kicks (Waring States Projects, 2007). Although this is a classic example it may not be perfectly clear that this distribution really is Poisson. Of course the number of deaths due to horse kicks can never be absolutely independent of the time when the statistical data was gathered. External factors like wars may have influenced the use of horses and therefore also the rate of deadly horse kicks. However it may still be valid to abstract from those details and just assume a process to satisfy the Poisson characteristics as this example shows. This is important for us since we may also encounter situations where we must ignore such dependencies for the sake of feasibility of our simulation.

3.4 Mapping from UML to Simulation

The classes Simulation, QueueProcess and ControlProcess, together with the javaSimulation framework form the basis for the executable performance model that we want to create from the design model of the SuD. Marzolla describes this process of mapping a design model into a performance model in great detail. However, since we want a simplified approach we do not just translate it but change it to match our needs.

In our approach it is possible to represent every element of the activity diagram with an element in the simulation. Table 1 shows the design model elements with their counterparts in the simulation.

3.4.1 Workload

As explained earlier, the workload defines the rate at which the first action of the scenario is called. Correspondingly, in the simulation the workload is implemented as a standard Process which repeatedly calls the first process of

Design Model	**Simulation**
Workload	Process
SimpleAction	QueueProcess
ForkAction JoinAction JunctionAction	ControlProcess

Table 1: Mapping of Design Model Element to Simulation Elements

the actual simulation with a certain interval time. Once started it just keeps running until the end of the simulation and does not have be activated again.

Listing 3: Workload

```
// definition of a process called "teller_serve"
Process workload = new Process()
{
   @Override
   protected void actions()
   {
      while (true)
      {
        // activate the first process of the actual simulation
        start_proces.activate();

        // wait some time
        hold(occurrence);
      }
   }
};
```

3.4.2 SimpleAction

SimpleAction is the only action that represents the actual processing of data in the program. Thus it requires a certain amount of time. Sequential calls to an action are stored in a queue which is processed sequentially. This behavior can be implemented with the QueueProcess class.

Listing 4: SimpleAction

```
// definition of a process representing a simpleAction
QueueProcess simple_action = new QueueProcess()
{
    @Override
    protected void loopAction()
    {
        for (int i=0; i < repetitions; i++)
        {
            hold(responseTime);

            if (i < (repetitions-1))
                hold(interval);
        }
        successor.activate();
    }
};
```

3.4.3 ForkAction

Concurrent executions of multiple steps are depicted as ForkActions in the activity diagram. These actions do not require time and start all successive actions, in parallel. To simulate this, the ControlProcess class is used.

Listing 5: ForkAction

```
// definition of a process representing a forkAction
ControlProcess fork_action = new ControlProcess()
{
    @Override
    protected void loopAction()
    {
        successor1.activate();
        successor2.activate();
        ...
        successorN.activate();
    }
};
```

3.4.4 JoinAction

Parallel threads of execution are joined in this actions. In this action the program waits for activation from each of its preceding actions before calling its successor. This behavior is implemented by recurring calls to passivate() until activate() was called exactly as many times as this action has predecessors.

Listing 6: JoinAction

```
// definition of a process representing a joinAction
ControlProcess join_action = new ControlProcess()
{
   @Override
   protected void loopAction()
   {
      for (int i=1; i < noOfPredecessors; i++)
         passivate();
      successor.activate();
   }
};
```

3.4.5 Junction

Conditional execution of actions is modeled with junctions. JunctionActions have arbitrarily many pairs of conditions and actions. If a condition holds true the corresponding action is called. In the simulation this is implemented using a series of if/else-if/else statements within a ControlProcess.

Listing 7: JunctionAction

```
// definition of a process representing a junctionAction
ControlProcess junction_action = new ControlProcess()
{
   @Override
   protected void loopAction()
   {
      if (condition1)
         successor1.activate();
      else if (condition2)
         successor2.activate();
      ...
      else (conditionN)
         successorN.activate();
   }
};
```

3.5 Analysis of the Simulation Results

Since an engineer is directly programming the simulation by hand it is easy for him to get all kinds of output data from the simulation. Whether he makes the program print out trace messages during the execution of the simulation, or he makes it write output data to a file, or he even makes it draw

a GUI representing the performance data visually is his decision.

The current simulated time is available via the static method time() of the class Process. The time that single process was active can be retrieved by calling getActiveTime() or getActivePercentage() on the particular process, depending on whether the absolute or the relative active time is needed.

Generally, the method report() is a fixed part of the Simulation class and shall be used for the reporting analysis results.

The average utilization of a process, represented by the relative active time can already be a good indicator that may present valuable information to an engineer as to where potential bottlenecks of the the SuD are and how the system may be balanced.

It is, however, impossible to tell anything about the utilization of a host that runs multiple processes in parallel. Since we do not consider resources explicitly, no connection can be made between different processes running on the same machine. The degree of resource sharing and blocking cannot be made explicit with this method.

Further on, the simplification of representing the queue of individual processes as a plain number, showing only the actual size of the queue makes the analysis less powerful when it comes to the analysis of waiting time and processing time of individual queued objects. We cannot say anything about how long an object waits in the queue before it is processed. As we said earlier this would require the representation of the queue as a real object queue. Also, the scheduling policy of the host that executes the process has a great influence on the waiting time of individual objects. The effect of different scheduling policies is impossible to show in this model.

4 Evaluation

It was our declared goal to present an approach to software performance engineering that (1) is correct, (2) is easy to learn and use, and (3) accounts for existing work that we can reuse in the analysis. The evaluation of our approach will mainly consist of a case study in which we will apply the performance analysis and see how useful it is there. The case study is a software project within the company Deutsche Börse Group.

The execution of the adapted performance engineering process is docu-

mented below, and difficulties that occurred to us during the application of the process are reported on. Eventually we will see in how far the results that we obtain are useful and whether spending the resources to conduct the analysis is worthwhile.

4.1 Case Study

The department of Knowledge Management within the Deutsche Börse Group is developing a web search engine for internal usage and for a major external customer. The search engine, called Xpider, is characterized by offering the possibility to specify search requests in great detail such that unwanted results can be filtered out very effectively. Furthermore Xpider always uses the latest available data, rather than returning previously cached and indexed websites. The web crawler which actually downloads the websites from the web servers on the Internet only starts crawling upon a user request.

Since crawling the Internet on request, rather than browsing through a previously filled database can be very time consuming, depending on the complexity of the request, the execution of a search can take up to several hours or even days. Additionally multiple requests may be processes at the same time.

With this search engine Deutsche Börse Group won the public tender of a German public authority. One of the most important arguments with which they won the tender was the great performance of that Xpider version. Given the special requirements, Xpider outperformed the competing products. Deutsche Börse was able to guarantee a total throughput of 100,000 crawled and filtered websites per day. Under laboratory conditions a peak value of 250,000 websites per hour has been measured. Recent analysis has shown that at the client's site approximately 1,000,000 websites are processed per day, on average.

Due to new functionality required by the customer, a new Xpider version (4.0) is currently being developed by the Knowledge Management team. While new features will be implemented into the new version the performance of the system shall be kept at a high level. Xpider 3.5 is a strongly multi-threaded and distributed system and the new version will be build to utilize threads and multiple hosts even better. In Xpider 4.0 several design decision have to be made. These include the choice of the right crawler, the network protocol, and the database design. Additionally the system's bottlenecks have to be identified for the developers to know which part of the

software limits the performance.
To make sure, from the very beginning, that the performance requirements are really met, an early performance analysis and prediction process has to be implemented. The goal is to be able to predict the performance of Xpider 4.0.

4.1.1 Problem Formulation

Xpider 4.0 is a redevelopment of Xpider 3.5, a component-based software developed by the Knowledge Management department at Deutsche Börse AG. Xpider is a web search engine used internally at DBAG and by an external customer. The software comprises a user front-end, a web crawler, a filter engine, a database management system, and a central server that links and manages those components.

In the new version the system's architecture shall be restructured to account for new functional requirements. Additionally, the central server and the web crawler will be newly developed.

With the simulation study the developers want to be able to make predictions about the performance of the Xpider 4.0. They are particularly interested in how many websites can be processed in a fixed amount of time. Furthermore they also want to know where the bottleneck in the system is, and thus where the software can be improved.

4.1.2 Objectives and Project Plan

The project is executed according to the procedure that was described in section 3.1. First, a model of the existing system is built. The behavior of the software is described in an activity diagram. At the same time performance data of the running system is collected. In particular the response times of selected components, the filter engine and the crawler are measured. These components were selected by the application's developers because they figured that their performance would determine the performance of the entire system.

After the model is built and the data is collected a simulation must be constructed. This is done by mapping each element of the UML activity diagram into a simulation process. The response times are simulated with the help of a random number generator which generates numbers according

to certain distribution functions. These distribution functions are fit to the data that was previously measured. This way the random generator class which is provided by the framework (cf. section 3.3.5) can be utilized to simulate the processing times.

The correctness of the simulation is validated by comparing the virtual throughput of the simulation to the actual throughput of the productive system. When the response times of the simulated components are correct and all important components and processes are represented the simulated and actual throughput should match.

Once the simulation of the existing system can be validated a new simulation is constructed which represents the planned system. This is done by adapting the existing system's model such that new features and properties of the new system are accounted for. What kind of adaptations are made is decided on after a first valid simulation is available and the developers have a better understanding of the system.

4.1.3 Model Conceptualization & Data Collection

Xpider 3.5 consists of the four basic components, a central server, crawlers ('workers'), filter engines ('catkits'), and a user front-end. Several worker and catkit instances can work in parallel. The fundamental crawling and filtering work-flow is as follows. The user triggers the process via the front-end. Correspondingly, the server orders the first worker to start crawling. After the download of the first website two actions are executed concurrently. On the one hand, new websites are enqueued for the crawlers to process. At the same time the previously crawled website is put in a second queue to be processed by the filter engine. Eventually the result is sent back to the front-end.

This behavior is shown in the activity diagram below. Different components are depicted in so-called swim-lanes, the columns in the diagram. The bubbles represent the different activities.The connecting arrows visualize the flow of control through the program. The black horizontal bar represents a fork. The two following actions are then running in parallel.

The multiple instances of the crawler and catkit are described in the comment box in plain text. Also the response times are informally shown in the box.

For the sake of simplicity the developers consider most of the above men-

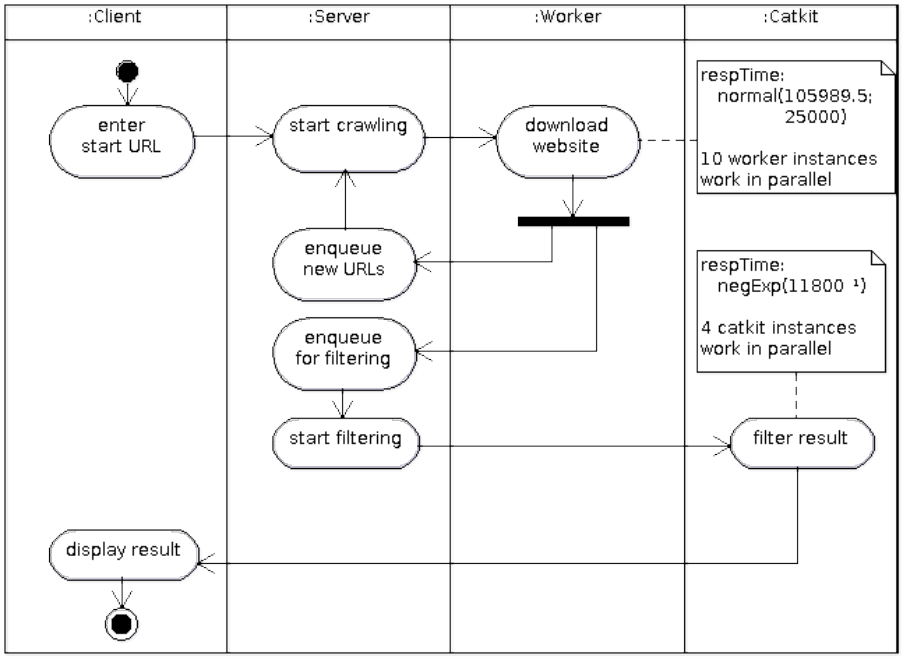


Figure 5: First Activity Diagram of Xpider 3.5

tioned activities to be instantaneous and thus not to require time at all. Only the response times of the crawling and filtering process are measured as those are assumed to be the determining factors for the performance of the system as a whole. The crawling and filtering durations per website are read from log files that are constantly printed out by the server.

In the diagram below (Figure 6), the response times of the catkits are depicted in a histogram. The height of the bars represents the number of observed response times that lay within the corresponding interval. Each interval, also called bin, comprises response times within a time span of one second. Hence most of the websites were filtered within one second. The second most websites were filtered in more than one second but less than two seconds. For very few websites the filtering process took more than 40 seconds to finish.

In figure 7 the response times of the crawlers are depicted in a similar fashion. The only difference is that the size of the bins is five seconds instead of one. However, the observations here look very different the measurements of the catkits' response times. In this case the most response times were

between 100 and 105 seconds. The other values surround this peak such that the diagram resembles a bell-shaped curve.

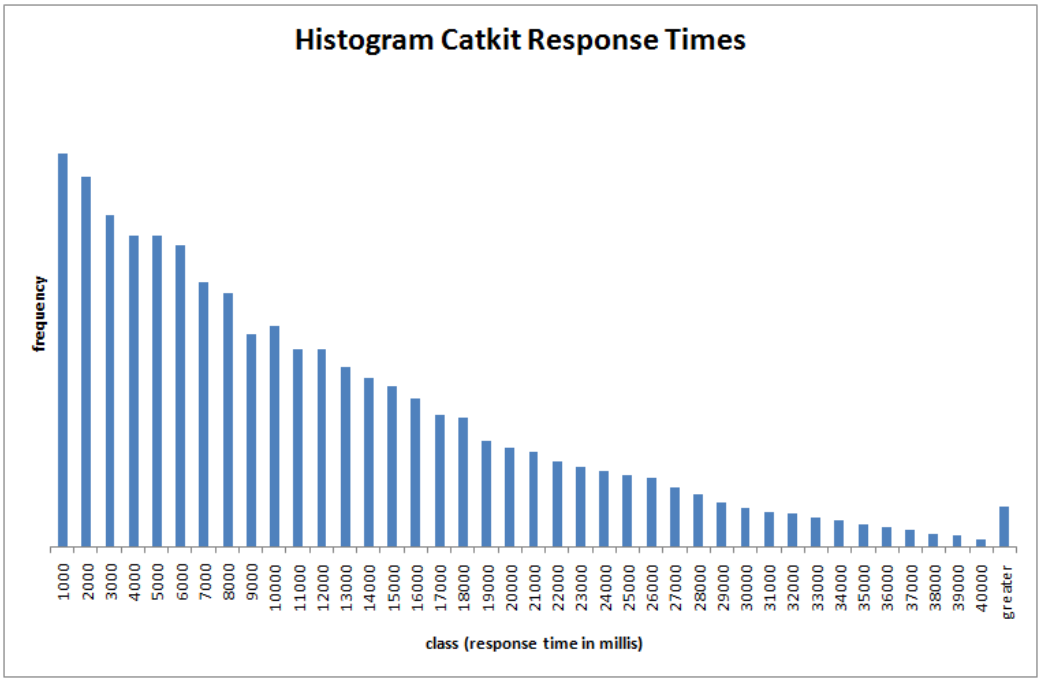


Figure 6: Frequencies of the response times of a catkit to filter one website.

4.1.4 Model Translation

In this step the activity diagram is translated into a simulation. This is done with the approach we proposed in section 3.4. For each element of the UML diagram there is one element in the simulation. Additionally the data that was collected in 4.1.3 is modeled in mathematical functions which are easy to handle for the simulation framework. The complete source code of the simulation is given in the appendix of this paper.

We started to implement the simulation for the scenario in which four catkit components work in parallel. The interface of the implementation is shown below. The complete source code of the simulation application is given in the appendix of the paper.

Within the simulation the response times of the two components catkit and worker were modeled as random values drawn from certain distribution functions. Those functions were selected such that they match the actually

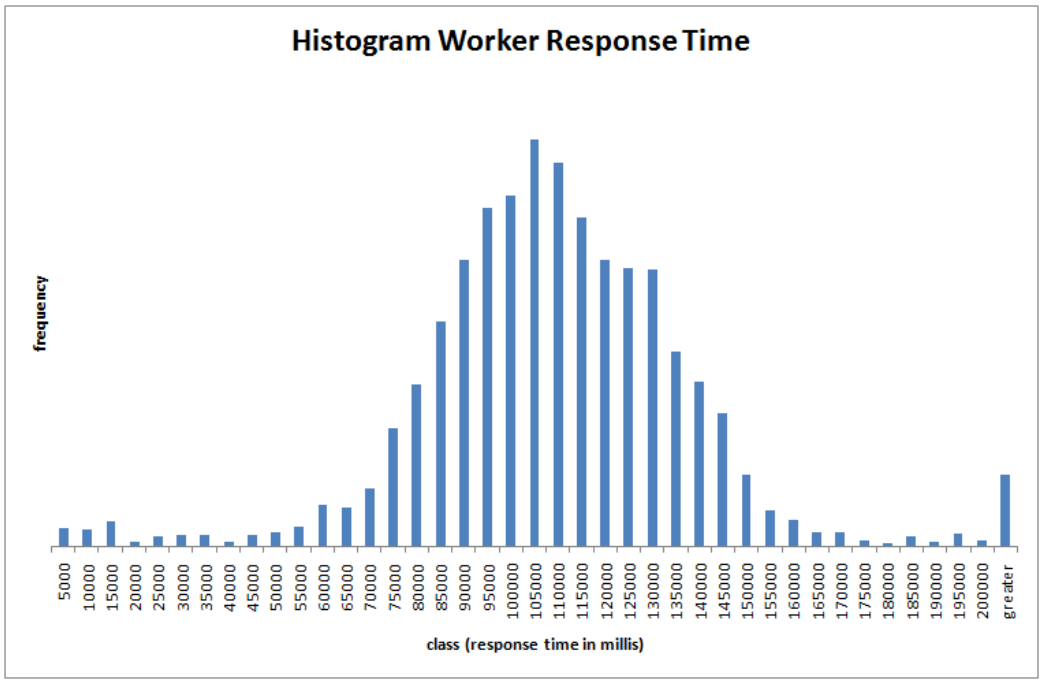


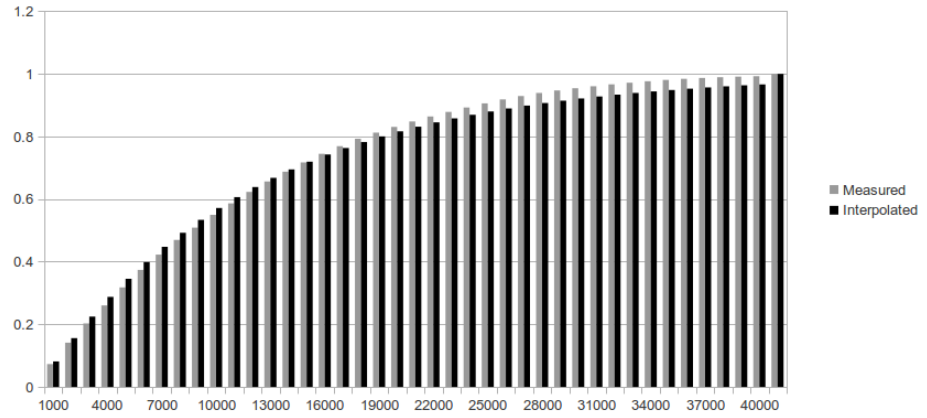
Figure 7: Frequencies of the response times of a worker to download one website.

measured data closely. The following diagrams show the histograms that were already presented in the previous section, but with cumulative frequencies (black). In the same diagrams those values are depicted that the model function return (gray). These functions are used in the simulation.

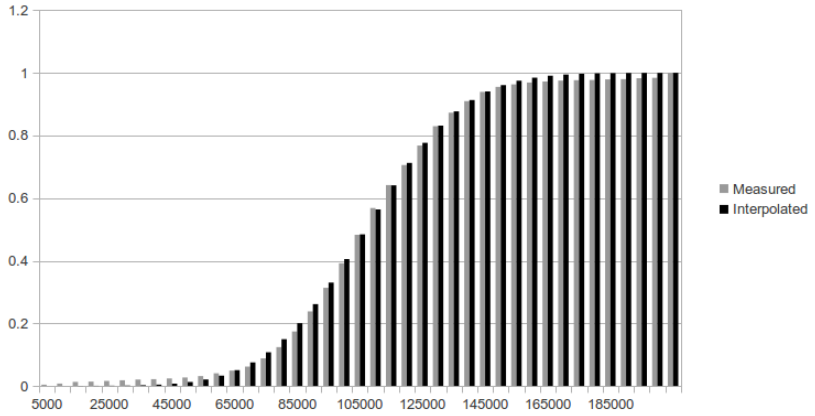
The normal distribution function with mean value 105989.5 and standard deviation 25000.0 and the negative exponential function with rate parameter 11800.0^{-1} for the crawlers' and the catkits' response times, respectively are good fits for the data. This is backed up by the fact that the Kolmogorov-Smirnov (K-S) goodness of fit test does not refute the hypotheses that the distributions match the data. The K-S test is conducted with a significance level of 0.05. In both cases the critical values are not exceeded. The K-S test is independent of the actual distribution and may be applied for both normal and exponential distribution in the same manner.

4.1.5 Verification & Validation

During the verification process the simulation program is reviewed to check whether it is really executable, and does not contain obvious errors. By using a framework and a, to some extend, standardized method to construct



(a) Response times of the catkit, interpolated by a negative exponential distribution with rate parameter $\lambda = 11800.0^{-1}$



(b) Response times of the crawler, interpolated by a normal distribution with mean $\mu = 105989.5$ and standard deviation $\sigma = 25000.0$

Figure 8: Cumulative distribution of measured and interpolated values.

the simulation, this is quite a simple step. In this paper we present detailed steps to translate the design model into a simulation. If the translation is executed as suggested there should not be errors in the code that prevent the simulation from running. In this case study the simulation was made executable within minutes and only minor errors like misspelled strings were fixed.

In contrast, the validation of a simulation is much more difficult. Validating a simulation means to check whether it is a good representation of the actual system.

Since we are only considering the time dependent behavior in this analysis, the simulation should show a behavior which is similar to that of the Xpider 3.5 software, in terms of response time and throughput.

In this case study we try to validate the by comparing the number of websites that are virtually processed by the simulation to the number of actually crawled and filtered websites in the real system, within one hour. The assumption behind this is that if all response times are correctly measured (that is the data is correct), and the all determining factors in the system have been found and represented (that is the model is correct), the hourly throughput of both applications should match.

Unfortunately, this is not the case. In 100 runs we did with our simulation, on average 339 websites were processed within one hour of simulated time. Whereas, the actual system processed more than 2000 real websites per hour. This significant difference indicates that the simulation is by no means a valid representation of the actual system.

4.1.6 Potential Error Sources and Solutions

There can be three potential sources of errors identified, which may be responsible for the failed validation: First, the measurements and therefore the data we used in the simulation may be wrong. We collected the response times from log-files that were printed out by the server. In these files, the start and end times of different processes are noted, such that the durations can be calculated. For this analysis, those calculations were done by a small program script. The reason for this being, that the size of the log-files usually exceeds one gigabyte and is therefore way too big to be manually analyzed. Of course, it is possible that the script's calculations and therefore the measured response times are wrong.

In fact, this explanation for the failed validation is not particularly convincing. The script and the calculations have been checked over and over again, also for their meaningfulness. Several values that the script returned, were manually double-checked.

Second, it is possible that the activity diagram does not represent all of the components and processes that are needed for a successful simulation of

this software. Maybe the project team's assumption, that the crawler and the filter engine are the only determining factors for the system's performance is wrong. It is possible that there are hidden processes that were not considered which actually do influence the overall performance of the system.

This could be a valid point, since the activity diagram is a great simplification of the actual system. Xpider 3.5 is a highly complex, multi-threaded and distributed software which grew in both functionality and size of the code. Even for the developers, estimating and modeling the performance related behavior is extremely difficult.

The third potential problem that we identified is that in the real system there exist interdependencies among components and processes which affect their performances, mainly due to shared resources. These dependencies could cause side-effects which are not accounted for in the simulation. For instance, we know that crawlers and filters are reading and writing from the same hard disk drives, so whenever these two components access the drive concurrently, one of them has to wait while the second one is served. Naturally, the waiting time is added to the processing time and increases the overall response time.

Because these dependencies and their effects are not obvious and not easily modeled, we purposefully tried to abstract from these details as explained in section 3.2. But, of course, this abstraction can also entail risks: The response times are taken for granted without knowing exactly why they are as long as they are, thus they cannot really be controlled; this means they cannot be reproduced when the exact causes are unknown. In turn this could mean that if small parameters in the scenario are changed, the abstractions and hence the simulation become invalid.

In this case study, the software is particularly dependent on the resources, such as the network and the hard disk drives. It may be possible that the response times that we measured are subject to strong variation, depending on the current network load, other jobs running on the systems and the actual file types that are processed. Accordingly, the response times and the throughput we measured would be nothing more than snapshots without generalizability. If this was the case, the discrepancy we see in the measured values would not come as a big surprise, since response times and throughput were measured in two separate runs for technical reasons.

One way of mitigating this last problem is to eliminate as many external factors in the system as possible. In the case study this was done by hosting a fixed set of websites on a local server which could then be crawled by the software. Yet, our simulation could not be validated with the given

data. Within the limited time frame of the project it was not possible to set up a new experiment where the system was running in complete isolation. Therefore, the system was still influenced by the network traffic on the local network, other crawl jobs that were running in the system and other processes that were running on the web server's machine.

We believe that without the set up of an isolated experiment, the creation of a valid simulation model is not feasible. That is considering that we do not want to represent resources and resource sharing in our model.

In other words we believe that in order to be able to create a valid simulation from an existing system without taking resources into account, the system must be tested and measured in an environment where all external factors are eliminated.

Since this was not possible during the case study, as we said above, the results from the measurements were taken into consideration while implementing Xpider 4.0, but no performance prediction was conducted.

4.2 Discussion

During the design of the framework and during its usage in the case study several positive aspects of the approach but also many weaknesses were discovered. Those points of criticism can be divided into three categories: On the one hand we have benefits and drawbacks of working with UML. On the other hand there are benefits and drawbacks of the simulation and the translation from UML to the simulation model. Furthermore we will comment on the reasonableness of the procedure that we described and applied, namely the approach adapted from Banks.

4.2.1 UML

The positive aspects of UML that we chose it for in the first place, have been confirmed. While discussing different models with the development team it was particularly easy to do that with the help of activity diagrams. However, during these discussions mostly quick sketches were used to depict the work flow within the application on the white board or a sheet of paper, which is why formally correct models were never used. This is particularly true for the performance annotations where response times were usually mere figures representing the mean value.

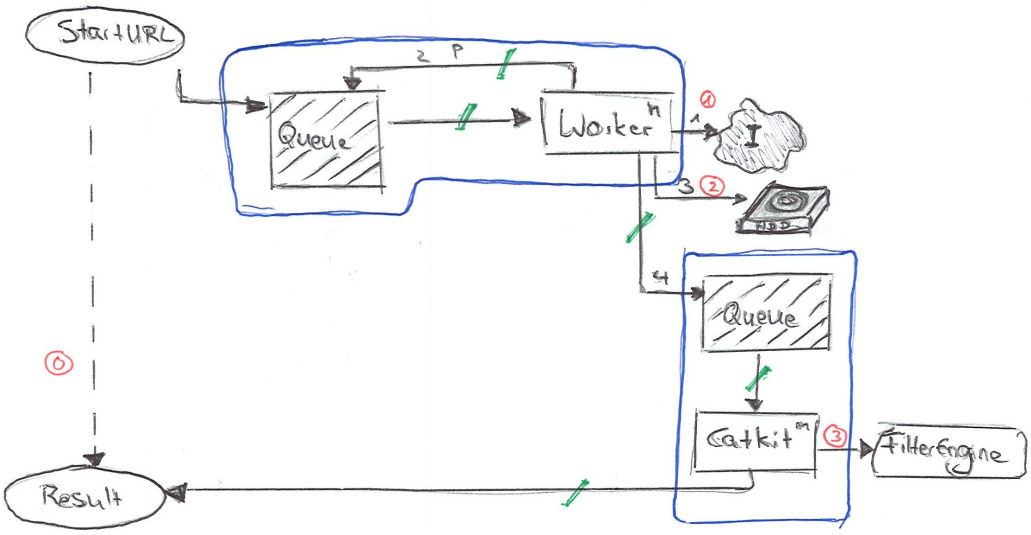


Figure 9: Draft model of the software as it is used internally by the developers.

However, especially in the first discussions in the project an informal notation was used by the developers that can be described as an intuitive way of representing software components, their communication and the object flow in the system in one diagram (cf. Figure 9). Due to the deliberate non-standard nature of the diagrams it is impossible to directly map those diagrams to a simulation model. Because of this fact often a three step approach was chosen: We translated the informal model into an activity diagram which could then be used to build the simulation. So even though the UML was not the preferred modeling language of the development team it served well as an intermediate step between the very informal representation and the executable simulation model.

One problem that we were facing while using UML was the difficult creation of diagrams in a diagramming tool. We used ArgoUML (http://argouml.tigris.org/) for this purpose since it is an open source tool and has an active community. Another advantage of ArgoUML is that it is written in Java and executes on the used Solaris platform ready to use without recompilation.

ArgoUML supports the creation of activity diagrams with all the model components that we described in section 3.2. Even though it is theoretically possible to draw the kind of diagram that we need we found the tool to be overly complex and not always intuitive. For example ArgoUML constantly checks for the well-formedness of the diagram which we found undesirable

and even confusing. Additionally we found redesigning of existing models to be too complicated such that we often ended up drawing the model from scratch and losing valuable time. These problems were specific to the tool that we used in this project, but they emphasize the fact that it may not be easy to find suitable tools to support the process.

A more general problem with UML as a modeling language was that it cannot represent multiplicity and replication of servers as we described it in section 3.3.4. This does not come as a surprise since the activity diagram is usually only concerned with what kind of activities are performed in the system but not how and where. The only exception here are so-called swim lanes. Swim lanes in UML diagrams can be used to group activities by the component or object that executes them. It may be possible to add information to swim lanes as to how many instances of a component are available. We did not further delve into that.

4.2.2 Simulation & Mapping

A major advantage of simulation as a performance model is its ability to demonstrate every behavior that the represented software shows. Theoretically the analyst does not have to make a compromise when simulating a software system. However, if the performance model is bound to the expressiveness of a design model this ability loses significance. This is, for instance, the case when the performance model is automatically generated from design model. Currently, in the approach we presented here the simulation is hand coded by the analyst. Usually it is based on a UML diagram but it may also extend the represented behavior. In the case study this has been done to simulate multiple Crawler and Catkit instances, which cannot be represented in an activity diagram (cf. previous section 4.2.1).

With the classes Simulation, QueueProcess and ControlProcess we provide convenience classes that may be used to base the simulations processes upon, but even so, these classes do not have to be used and may be mixed with the standard base classes from the javaSimulation framework. Therefore our framework provides both, a basis that allows for the convenient and fast creation of software simulations and the full functionality of the javaSimulation framework that may be used to create simulations of virtually unlimited complexity.

An additional advantage of the manual translation from the design to

the executable model is that the model is double-checked and interpreted by a human-being. This is particularly helpful when shorthand notations and sketches are used rather than standard-compliant activity diagrams, as explained above.

However, the mere design of the simulations "join process" as well as the case study reveals one significant weakness that this framework displays: Simulated processes have no notion of their caller. In other words a process does not see and therefore cannot distinguish between different processes that trigger its execution.

This has two major effects that prevent this framework from being used in all but the most trivial scenarios.

- The "join process" only counts its activations but does not really know whether those activations came from the processes that actually ought to be joined. This is particularly problematic when processes that must be joined have very different response times.
- "Blocking processes" cannot be realized in the given system. By blocking we mean processes that other processes rely on and wait for until they have finished their execution. Marzolla describes those processes that rely on others to finish first "composite steps" (as opposed to simple steps).
  Here the problem is that the dependent step may not proceed with its execution before the processes it relies on finish. Hence it has to wait for an activation command from exactly that process. Unfortunately there is currently no way of telling the origin of an activate() call.
  The issue worsens when one blocking process is used by multiple other processes. In that case the framework or the blocking process itself also has to keep track of the order in which dependent processes have to be reactivated.

From this problem it follows that only systems with strictly linear behavior can be represented. Linearity in this context means that the performance of a system is independent of the workload, i.e. the response times of individual processes is always constant. In practice this would mean that, for example, by doubling the number of servers the throughput of the system would also double. Due to other restrictions, for example the network bandwidth which caps the throughput, this can never be true.

4.2.3 Procedure

The problems that we encountered in the case study make it impossible for us to give a comprehensive experience report on the approach we applied here. In the project the procedure was stopped with the validation of the base system's model. In the project it was not possible to create a valid simulation model of the existing system within an acceptable time-frame. However, the fact that we actually had these major problems raised the following critical points.

Firstly, it shows the high significance of a structured performance analysis in software development in general. The engineers of the Xpider software were not able to draw absolutely clear conclusions from the analysis that was undertaken, even though they knew the system's architecture and its behaviors for several years. In similarly complex or even more complex systems it can be practicably impossible to understand and improve a software's performance without a thorough analysis.

Secondly, from the experience we made during the Xpider project we presume that even if it should be possible to create a validated model from the base system, the next step, namely deriving a model for a non-existent system is yet harder to accomplish. In turn this emphasizes the necessity of using a structured and comprehensive approach for achieving a useful prediction. Our initial thesis that the validation of whether the used model really takes the important and potentially limiting factors and components into consideration must not be underestimated was supported, firmly. Therefore the approach of backing up a new model with a validation based on hard facts and measurements is certainly not only helpful but absolutely necessary according to our experience.

Thirdly, and this is a major point of criticism, the approach is not explicit and comprehensive enough to be applied in real projects as is. The focus of the approach to derive a model from a previously created one may be fair and important but it also makes the approach hard to use without additional guidelines, for instance on how to interpret data that was gathered from a running system.

All in all, for the approach that was applied we can say that the case study proved the necessity of such a structured procedure for the performance analysis of a complex software system. Also the strong focus on the validation of the model in order to achieve meaningful results turned out to be important

since we were not even able to create a valid model of an existing system, not to mention a model of system that has yet to be built. But because of this narrow focus this approach can only be used in conjunction with other guidelines that help in executing steps like the data collection and the model building.

5 Summary & Future Work

The goal of this paper is to present an approach to performance engineering in the area of software development. The approach should be particularly suitable for the application in early stages of the development life-cycle. The outcome of the approach should be a model which can be used for performance predictions on a future software system. The accuracy of the model and the therefore the prediction shall be ensured by basing in on an existing base system's model.

The general approach is an adapted version of the guideline by Banks et al. (2001). This adaption has a strong focus on the validation of the system's model, thus the check of whether the model is a good representation of the real system. More precisely, in the presented approach the model of the system which is under development is based on a model of an existing and comparable system, the so-called base system. Therefore the potentially difficult validation of a model whose real counterpart does not yet exist is avoided since the model is based on another model that can be validated relatively easily.

The approach involves the creation of a so-called design model that represents the system's characteristics in a human-readable and intuitive way. In a second phase this design model is translated into a performance or operational model. This, in turn, should be presented in a machine-readable format such that a computer program can read and execute it.

In our approach we chose the UML activity diagrams as the design model, since such diagrams are commonly used among computer specialists which makes them quite intuitive and easy to use. Activity diagrams describe the behavior of a system but not are not capable of representing its structure. We deliberately chose for this restriction as we wanted to make the approach particularly easy and did not want to burden the users with details about the underlying physical architecture of the system. We chose simulation as our operational model since we found it to be both powerful and intuitive to use.

For the implementation of the simulation we suggest using the javaSimulation framework, a library of Java classes that provides a solid base for the construction of simulations with Java. We developed a few extensions to javaSimulation and provided a translation approach to build a simulation according to a UML design model. The translation consists of a mapping from the different elements of the activity diagram, annotated with performance parameters, to Java objects in the simulation program. Our extensions ease the translation by implementing common functionality on top of the used framework. Additionally, they provide functionality to get the busy time of the processes they represent. If other analysis data is required, instances of the base classes may override and extend this functionality as the user desires.

The procedure, including the modeling and the simulation was applied in a case study within a project at the Deutsche Börse AG, Frankfurt am Main, Germany. In the case study no valid representation of the existing system could be created. We identified three potential reasons, why the validation could not be successfully conducted. This led to the conclusion that under the given circumstances in the experiment, and the limited time-frame no reasonable and valid measurement and validation could be achieved. We strongly suspect that in a more controlled environment where the system runs in isolation, a valid representation and simulation may have been possible.

Despite the fact that the case study could not be conducted as desired it demonstrated quite impressively how important a structured approach in performance engineering is. In this project we failed to created a valid base model representing a productive system. If no validation of this base system would have taken place we may have ended up with meaningless, wrong and potentially deceptive predictions.

The next steps in this project would be to enhance the framework and extend the provided functionality to overcome the problems which are stated in the discussion section. With these enhancements a new attempt should be undertaken to actually apply the approach in a real project. Only after the successful application of the framework its practicability can be evaluated.

Bibliography

Abdullatif, A.A. & Pooley, R. (2008). *A Computer Assisted State Marking Method For Extracting Performance Models From Design Models.* International Journal of Simulation Systems, Science & Technology, UK.

Abdullatif, A.A. & Pooley, R. (2009). *From UML to EQN: Studying System Performance from an early Stage of System Life Cycle.* 25th UK Performance Engineering Workshop, Leeds, UK.

ArgoUML. Open source UML Modeling Tool. http://argouml.tigris.org/

Arief, L.B. & Speirs, N.A. (2001). *A UML Tool for an Automatic Generation of Simulation Programs.* Proceedings of the 2nd international Workshop on Software and Performance, New York, NY, USA.

Arlitt, M.F. & Williamsson, C.L. (1997). *Internet Web Servers: Workload Characterizations and Performance Implications.* IEEE/ACM Transactions of Networking (TON), Vol. 5, No. 5, IEEE Press Piscataway, NJ, USA.

Balsamo, S. et al. (2004). *Model-Based Performance Prediction in Software Development: A Survey.* IEEE Transactions of Software Engineering, Vol. 30, No. 5.

Banks, J. (1999). *Discrete Event Simulation.* Proceedings of the 1999 Winter Simulation Conference, p. 7-13, Marietta, Georgia.

Banks, J. et al. (2001). *Discrete-Event System Simulation.* Prentice Hall, Upper Saddle River, NJ, USA.

Bass, L., Clements, P. & Kazman, R. (1998). *Software Architecture in Practice.* SEI Series in Software Engineering.

Bell, D. (2003). *UML basics: An introduction to the Unified Modeling Language.* IBM DeveloperWorks. Retrieved 04/23/2012 from http://www.ibm.com/developerworks/rational/library/769.html.

Bosch, J. & Molin, P. (1999). *Software Architecture Design: Evaluation and Transformation.* Proc. 1999 IEEE Eng. of Computer Systems Symp, pp. 4-10.

Briand, L.C., Labiche, Y. & Leduc, J. (2004). *Towards the Reverse Engineering of UML Sequence Diagrams for Distributed, Multithreaded Java Software*. IEEE Transactions om Software Engineering, Vol. 32, No. 9.

Bruck, R. (2006). *Poisson and Exponential: Connected*. University of Southern California. Retrieved 04/20/2012 from http://imperator.usc.edu/~bruck/classes/fall2006/PoissonAndExponential.pdf

Conway, M. E. (1963). *Design of a Separable Transition-Diagram Compiler*. Communication of the ACM, Vol. 6, No. 7, pp. 396-408. New York, NY, USA.

Dahl, O.J. & Nygaard, K. (1966). *SIMULA: an ALGOL-based Simulation language*. Communications of the ACM, Vol. 9, No. 9, New York, NY, USA.

Dobing, P. & Parsons, J. (2006). *How UML is used*. Communications of the ACM - Two decades of the language-action perspective, Vol. 49, No. 5, New York, NY, USA.

Fagan, M.E. (1976). *Design and Code inspections to reduce errors in program development*. IBM Systems Journal, Vol. 15, No. 3, pp. 182-211.

Franks, G. et al. (2012). *Layered Queueing Network Solver and Simulator User Manual*. Department of Systems and Computer Engineering, Carlton University, Ottawa, ON, Canada.

Helsgaun, K. (2000). *Discrete Event Simulation in Java*. Department of Computer Science. Roskilde University, Denmark.

Helsgaun, K. (2001). *jDisco - a Java package for combined discrete and continuous simulation*. Department of Computer Science. Roskilde University, Denmark.

Law, A.M. (2003). *How to Conduct a successful Simulation Study*. Proceedings of the 2003 Winter Simulation Conference, Tucson.

Marzolla, M. (2004). *Simulation-Based Performance Modeling of UML Software Architectures*. Università Ca' Foscari Di Venezia.

Matousek, M. & Schneider, J. (1976). *Untersuchungen zur Struktur des Sicherheitsproblems von Bauwerken*. Bericht/Institut für Baustatistik und Konstruktion, ETH Zürich, Nr. 59, Zürich.

Object Management Group (OMG). (1997). *UML Specification 1.1.* OMG document ad/97-08-11. Omg.org.

Object Management Group (OMG). (2002). *UML^{TM}Profile for Schedulability, Performance, and Time Specification.* Version 1.1. formal/05-01-02.

Pooley, R. (2000). *Software Engineering and Performance: A Road-map.* ICSE 2000, Proceedings of the Conference on the Future of Software Engineering, Limerick, Ireland.

Shannon, R.E. (1998). *Introduction to the Art and Science of Simulation.* Proceedings of the 1998 Winter Simulation Conference, Texas.

Smith, C.U. & Woodside, M. (1999). *Performance Validation at Early Stages of the Software Development.* System Performance Evaluation. CRC Press, Inc., Boca Raton, FL, USA.

Smith, C.U. & Williams, L.G. (2001). *Performance Engineering: A Practical Guide to Creating Responsive, Scalable Software.* Pearson Education, Inc., Indianapolis, IN, USA.

Smith, C.U. & Williams, L.G. (2002). *PASA^{SM} : A Method for the Performance Assessment of Software Architecture.* Workshop on Software and Performance '02, Rome, Italy.

Starfield, T. (2011). *Modeling Basics: Rapid Prototyping Applied to Modeling.* University of Vermont, VT, USA. Retrieved 05/23/2012 from http://www.uvm.edu/~ tdonovan/modeling/Module2/ 02_RapidPrototyping_transcript.pdf

Waring States Project (2007). *The Poisson Distribution.* University of Massachusetts, Amherst, MA, USA. Retrieved 04/23/2012 from http://www.umass.edu/wsp/statistics/lessons/poisson/index.html.

Appendices

Source Code of the Xpider Simulation Case Study

Listing 8: Complete Source Code of the Case Study

```
package de.exchange.exotic.xpidersim;

import de.exchange.exotic.xpidersim.base.ControlProcess;
import de.exchange.exotic.xpidersim.base.QueueProcess;
import javaSimulation.*;
import javaSimulation.Process;

public class Simulation extends Process
{
   Random r = new Random();

   final double averageCrawlTimePerWebsite = 105989.5;
   final double standDeCrawlTimePerWebsite = 25000.0;

   final double averageFilterTimePerWebsite = 11800.0;

   Process workload = new Process()
   {
      @Override
      protected void actions()
      {
         client_enter_starturl.activate();
      }
   };

   QueueProcess client_enter_starturl = new QueueProcess(
         "client_enter_starturl")
   {
      @Override
      protected void loopAction()
      {
         server_start_crawling.activate();
      }
   };

   QueueProcess server_start_crawling = new QueueProcess(
         "server_start_crawling")
   {
      @Override
```

```
    protected void loopAction()
    {
        worker_download_website.activate();
    }
};

QueueProcess worker_download_website = new QueueProcess(
        "worker_download_website")
{
    int lfdNummer = 0;

    final int noOfWorkers = 10;
    Head workersAvailable = new Head();
    Head workersBusy = new Head();

    class WorkerProcess extends ControlProcess
    {
        public WorkerProcess()
        {
            super("worker" + lfdNummer++);
        }

        @Override
        protected void loopAction()
        {
            into(workersBusy);

            double respTime = r.normal(averageCrawlTimePerWebsite
                , standDeCrawlTimePerWebsite);
            hold(respTime);

            worker_fork_after_download.activate();

            into(workersAvailable);
        }
    }

    int availableCardinal = 0;
    int busyCardinal = 0;

    @Override
    protected void loopAction()
    {

        while (workersAvailable.cardinal() <= 0)
        {
            availableCardinal = workersAvailable.cardinal();
            busyCardinal = workersBusy.cardinal();
```

```
            if ((availableCardinal + busyCardinal) < noOfWorkers)
               new WorkerProcess().into(workersAvailable);
            else
               hold(1);
         }

         ((ControlProcess) workersAvailable.first()).activate();
      }
   };

   ControlProcess worker_fork_after_download = new ControlProcess(
         "worker_fork_after_download")
   {
      @Override
      protected void loopAction()
      {
         server_enqueue_for_filtering.activate();

         server_enqueue_new_urls.enque(4);
         server_enqueue_new_urls.activate();
      }
   };

   QueueProcess server_enqueue_new_urls = new QueueProcess(
         "server_enqueue_new_urls")
   {
      @Override
      protected void loopAction()
      {
         server_start_crawling.activate();
      }
   };

   QueueProcess server_enqueue_for_filtering = new QueueProcess(
         "server_enqueue_for_filtering")
   {
      @Override
      protected void loopAction()
      {
         server_start_filtering.activate();
      }
   };

   QueueProcess server_start_filtering = new QueueProcess(
         "server_start_filtering")
   {
      @Override
      protected void loopAction()
      {
```

```
        catkit_apply_filter.activate();
    }
};

QueueProcess catkit_apply_filter = new QueueProcess(
        "catkit_apply_filter")
{
    int lfdNummer = 0;

    final int noOfWorkers = 4;
    Head workersAvailable = new Head();
    Head workersBusy = new Head();

    class WorkerProcess extends ControlProcess
    {
        public WorkerProcess()
        {
            super("catkit_worker" + lfdNummer++);
        }

        @Override
        protected void loopAction()
        {
            into(workersBusy);

            hold(r.negexp(1.0 / averageFilterTimePerWebsite));
            client_display_result.activate();

            into(workersAvailable);
        }
    }

    int availableCardinal = 0;
    int busyCardinal = 0;

    @Override
    protected void loopAction()
    {

        while (workersAvailable.cardinal() <= 0)
        {
            availableCardinal = workersAvailable.cardinal();
            busyCardinal = workersBusy.cardinal();

            if ((availableCardinal + busyCardinal) < noOfWorkers)
                new WorkerProcess().into(workersAvailable);
            else
                hold(1);
        }
```

```
        ((ControlProcess) workersAvailable.first()).activate();
    }
};

QueueProcess client_display_result = new QueueProcess(
        "client_display_result")
{
    @Override
    protected void loopAction()
    {
        completed++;
    }
};

protected void printExec(String name)
{
    System.out.println("Executed at " + time() + ":" + name);
}

int completedSum = 0;
int completed = 0;
int runs = 100;

@Override
protected void actions()
{
    for (int i = 0; i < runs; i++)
    {
        activate(workload);
        hold(3600000);

        report();

        completedSum += completed;
        completed = 0;
    }

    finalreport();
}

private void report()
{
    System.out.println(completed
            + " websites have been crawled and filtered!");
    System.out.println("Worker                was busy "
            + worker_download_website.getActivePercentage()
            + "% of the time.");
    System.out.println("Catkit Filter         was busy "
```

```
            + catkit_apply_filter.getActivePercentage()
            + "% of the time.");

    }

    private void finalreport()
    {
        System.out.println("On average " + completedSum / runs
                + " websites have been crawled and filtered!");
    }

    public static void main(String[] args)
    {
        activate(new Simulation());
    }
}
```