An Approach for Performance Evaluation of Software Architectures using Prototyping

Frans Mårtensson, Håkan Grahn, and Michael Mattsson

Department of Software Engineering and Computer Science
Blekinge Institute of Technology
P.O. Box 520, SE-372 25 Ronneby, Sweden
{Frans.Martensson, Hakan.Grahn, Michael.Mattsson}@bth.se

ABSTRACT

The fundamental structure of a software system is referred to as the software architecture. Researchers have identified that the quality attributes of a software system, e.g., performance and maintainability, often are restricted by the architecture. Therefore, it is important to evaluate the quality properties of a system already during architectural design. In this paper we propose an approach for evaluating the performance of a software architecture using architectural prototyping. As a part of the approach we have developed an evaluation support framework. We also show the applicability of the approach and evaluate it using a case study of a distributed software system for automated guided vehicles.

KEYWORDS: Software architecture, performance evaluation, architecture prototyping, architecture evaluation.

1. Introduction

The size and complexity of software systems are constantly increasing. During recent years, software engineering research has identified that the quality properties of software systems, e.g., performance and maintenance, often are constrained by their architecture [1]. Before committing to a particular software architecture, it is important to make sure that it handles all the requirements that are put upon it, and that it does this reasonably well. Bad architecture design decisions can result in a system with undesired characteristics, e.g., low performance and/or low maintainability.

When designing an architecture, there exists many different solutions to a given problem. Therefore, the design of the architecture should be supported by a well-defined, explicit method and reliable data predicting the effects of design decisions, preferably in a quantifiable way. Examples of architecture evaluation methods are prototyping and scenario-based evaluation [2]. Each method has its own advantages and drawbacks, and there is no general consensus that a certain method is the best. Which method to use depends on time constraints and which quality attributes that are to be evaluated.

One important quality attribute to evaluate during architectural design is performance. Many times performance problems are not detected until system integration test, and thus are very costly to correct [3]. Some even argue that a design change is at least ten times more expensive after the code has been written than during architectural design. Therefore, it is important to evaluate the performance of a system as early as possible in the system development process, i.e., during the architectural design phase.

In this paper, we present an approach that assess the performance characteristics of a software architecture, or a part of it. We apply the approach in a case study, an automated guided vehicle (AGV) system, where an early version of a communication component of the architecture is evaluated in order to identify its performance characteristics. The protypical method is based on an adaptation of the simulation based evaluation method as described by Bosch in [2]. We extend that approach by building an executable prototype of the software architecture, and thus evaluate the performance at the architectural level. The extensions to Bosch's method include, among others, the introduction of an evaluation support framework for gathering data in a consistent way during several subsequent evaluations as well as evaluation of candidate implementations or technologies.

We will with some background about software architecture in Section 2. In Section 3, we describe the simulation-based evaluation method, how we adapted it to prototype-based evaluation, and finally describe the resulting evaluation approach. Then, in Section 4, we illustrate the prototype based evaluation approach using a case study where an evaluation is conducted on an AGV system architecture. In Section 5 and Section 6 we discuss the results of the case study and how the evaluation approach worked, respectively. Finally, we conclude our study in Section 8.

2. Software Architecture

Software systems are constructed with a requirement specification as a base. The requirements in the requirement specification can be categorized into *functional* requirements and *non-functional* requirements, also called *quality requirements*. The design of software systems has traditionally been centred around the functional requirements. Although software engineering practice was forced to incorporate the quality requirements as well, software engineering research focused on the system functionality.

During recent years, the domain of software architecture [1, 4, 5] has emerged as an important area of research in software engineering. This is in response to the recognition that the architecture of a software system often constrains the quality attributes. Thus, architectures have theoretical and practical limits for quality attributes that may cause the quality requirements not to be fulfilled. If no analysis is done during architectural design, the design may be implemented with the intention to measure the quality attributes and optimize the system at a later state. However, the architecture of a software system is fundamental to its structure and cannot be changed without affecting virtually all components and, consequently, considerable effort.

Software architecture can be divided into three problem areas, i.e., designing, describing, and evaluating a software architecture. In this paper we focus on evaluating software architectures, and in particular evaluating their performance. Four approaches to architecture evaluation can be identified, i.e., scenarios, simulation, mathematical modelling, and experience-based reasoning. Smith [3] discusses an approach to modelling system performance mathematically, although one may require simulation in certain cases. Our approach relies on the construction of an executable prototype of the architecture.

3. The prototype-based evaluation approach

In the core of the prototype-based evaluation approach is the architecture prototype that approximates the behavior of the completed software system. When we were asked to perform the evaluation of the AGV system (see Section 4), we were unable to find a description of the steps involved in creating an architecture prototype. As a result we decided to take the basic workflow from simulation based evaluation as described in [2] and adapt it to our needs. We will in the following section give a short introduction to the steps involved in performing a simulation-based architecture evaluation. We will then describe the changes that we made to that approach, and finally describe the resulting prototype-based evaluation approach.

3.1. Simulation based architecture evaluation

A simulation-based evaluation is performed in five steps [2]:

1. Define and implement context.
2. Implement architectural components.
3. Implement profile.
4. Simulate system and initiate profile.
5. Predict quality attribute.

Define and implement context. In this first step two things are done. First, the environment that the simulated architecture is going to interact with is defined. Second, the abstraction level that the simulation environment is to be implemented at is defined (high abstraction gives less detailed data, low abstraction gives accurate data but increases model complexity).

Implement architectural components. In this step the components that make up the architecture are implemented. The component definitions and how the components interact with each other can be taken directly from the architecture documentation. The level of detail and effort that is spent on implementing the architecture components depends on both which quality attribute that we are trying to evaluate and the abstraction level that we have chosen to conduct the simulation at. If we are going to evaluate several quality attributes, then we will most likely have to implement more functionality than if we focus on only one.

Implement profile. A profile is a collection of scenarios that are designed to test a specific quality attribute. The scenarios are similar to use-cases in that they describe a typical sequence of events. These sequences are implemented using the architectural components that are going to be evaluated. This results in a model of the system components and their behavior. How a profile is implemented depends on which quality attribute we are trying to assess as well as the abstraction level that is necessary for getting relevant data.

Simulate system and initiate profile. The simulation model is executed. During the execution data is gathered and stored for analysis. The type of data that is gathered depends on which quality attribute that we want to evaluate.

Predict quality attribute. The final step is to analyse the collected data and try to predict how well the architecture fulfills the quality requirements that we are trying to evaluate. This step is preferably automated since a simulation run usually results in a large amount of raw data.

3.2. Adaptations to the evaluation method

The workflow from the simulation-based evaluation had to be adapted to incorporate steps that we wanted to perform in our prototype-based evaluation. The main changes that we made were to introduce an evaluation support framework and put more emphasis on iteration in the evaluation process. We also did minor changes in the existing steps. These changes are described in more detail when we present our case study.

3.2.1. Evaluation support framework

We added the step of creating an evaluation support framework for use during the evaluation. A layered view of where the support framework is placed is shown in Figure 1. We choose to create the evaluation support framework for two reasons.

First, it makes us less dependent on the architecture component that we want to evaluate. The framework decouples the architecture component that we are evaluating from the architecture model that is used to generate input to the component. This increases the reuseability of the architecture model as it only depends on the API provided by the framework and not directly on the architecture component.

Second, all logging can be performed by the framework, resulting in that neither the architecture model nor the architecture component that are evaluated need to care about the logging. This leads to both that the logging is done in a consistent way independent of the underlying architecture component, and that no change has to be made to the architecture component when it is fitted to the framework. All that is needed is a wrapper class that translates between the calls from the framework and the architecture component. A more thorough discussion on how we constructed our framework can be found in section 4.2.

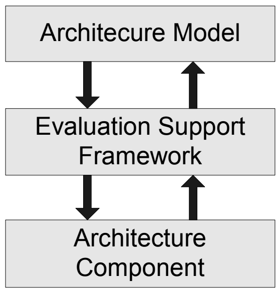

Figure 1: A layered view of the prototype design.

Iteration

During the development and execution of the prototype we found that it became necessary to perform the development of both the architecture model and the evaluation support framework in an iterative way. We needed to reiterate steps two to five in order to make adjustments to the way data was logged, and also to the behavior of the architecture model that was used. The need to make these changes was identified first after an initial execution of the simulation and analysis of the generated data. The positive thing with adding an iteration is that the initial results can be reviewed by experts (if such are available) that can determine if the results are sensible or not, and if changes to the model are necessary. We also got a confirmation that the log analysis tools were working correctly.

3.3. Prototype based architecture evaluation

In order to perform a prototype based evaluation there are some conditions that has to be fulfilled.

- First, there has to be at least one architecture defined, if the goal of the evaluation is to compare alternative architectures to each other then we will of course need more.
- Second, if we want to evaluate the performance of one or more candidates for a part of the software architecture then these components has to be available. This is usually no problem with COTS components but might pose a problem if the components are to be developed in house.

In addition, it is a preferable, but not necessary condition, that the target platform (or equivalent) of the architecture is available. If it is possible to run the prototype on the correct hardware, it will give more accurate results.

After integrating our adaptations in the evaluation method we ended up with the following method for prototype based architecture evaluation.

1. Define evaluation goal.
2. Implement an evaluation support framework.
3. Integrate architectural components.
4. Implement architecture model.
5. Execute prototype.
6. Analyse logs.
7. Predict quality attribute.
8. If necessary, reiterate.

Define evaluation goal. Define what it is that should be evaluated, are we looking at more one or more architecture candidates or architecture components, and which quality attributes are we interested in evaluating.

Implement an evaluation support framework. The evaluation support framework's main task is to gather data that is relevant for fulfilling the evaluation goal that has been defined. Depending on the goal of the evaluation, the framework has to be designed accordingly, but the main task of the support framework is always to gather data. The support framework can also be used to provide common functions such as utility classes for the architecture model.

Integrate architectural components. The component of the architecture that we want to evaluate has to be adapted so that the evaluation support framework can interact with it.

Implement architecture model. Implement a model of the architecture with the help of the evaluation support framework. The model together with the evaluation support framework and the component that is evaluated becomes an executable prototype.

Execute prototype. Execute the prototype and gather the data for analysis in the next step. Make sure that the execution environment matches the target environment as close as possible.

Analyse logs. Analyse the gathered logs and extract information regarding the quality attributes that are under evaluation. The analysis is with advantage automated as much as possible since the amount of data easily becomes overwhelming.

Predict quality attribute. Predict the quality attributes that are to be evaluated based on the information from the analysed logs.

If necessary, reiterate. This goes for all the steps in the evaluation approach. As the different steps are completed it is easy to see things that were overlooked during the previous step or steps. Once all the steps has been completed and results from the analysis are available, you could let an expert review them and use the feedback for deciding if adjustments have to be done to the prototype. These adjustments can be necessary in both the architecture model and the evaluation support framework. Another advantage is that it is possible to make a test run to validate that the analysis tools are working correctly and that the data that is gathered really is useful for addressing the goals of the evaluation.

4. A case study of an AGV system

The prototype based evaluation approach was specified in order to perform an evaluation for Danaher Motion Särö AB [6] that is developing a new version of a control system for Automated Guided Vehicles (AGV:s). The system consists of a central server that controls a number of vehicles through a wireless network. Each vehicle has a client that controls it and communicates with the server. The client regularly position the vehicle through, e.g., laser measurements. The position is then sent back to the server which, based on the positioning information and information stored in a map database, decides what the client is to do next. Typical commands for the client is to drive a certain sequence of path segments, or load and unload cargo.

The client in the new system has a number of quality requirements that has to be accommodated, for example portability between different operating systems, scalability in functionality, and cost efficiency. The cost efficiency of the client is largely influenced by the price of the hardware that is needed to provide the necessary processing power to complete its computational tasks within a given timeperiod. This brings us to the performance of the client, since an efficient client will be able to work on slower hardware than a less efficient version, i.e., a more efficient client will be more cost efficient. The target platform for the new client is a Intel Pentium CPU at 133 MHz with an embedded version of the Linux operating system.

The prototype based evaluation method is applied to the architecture of the client and focus on how the internal communication in the client is handled. The clients consist of a number components that exchange information with each other. The components are realised as a number of threads within a process. In order to decrease the coupling between the components it was decided to introduce a component that provided a level of indirection between the other components by managing all communication in the client. This communication component is very crucial for the overall performance of the client as all communication between the other components in the client goes through this component. The communication component handles asynchronous communication only, the components communicate with each other by publishing telegrams (messages) of different types. Each component that is interested in some type of information has to register as a subscriber for that telegram type.

A first version of the communication component was already in use for the development of the other components. There were however some concerns regarding how well the communication component would perform on the target hardware for the new system. In order to verify that the new client would be able to fulfill the performance requirement it was decided that a performance evaluation should be done before to much time was spent on further development.

We will now go through the steps in the prototype-based evaluation method and describe what we did in each step. This will hopefully give the reader a more concrete feeling for the tasks that have to be done.

4.1. Define the evaluation goal

We defined the goal of the evaluation to be the performance of the communication component of the new AGV client. The component is critical as it handles the dispatching of messages between all the components in the client.

Because of safety and requirements regarding the positioning accuracy of the vehicles, the client has to complete a navigation loop within 50 ms. During this time a number of messages has to be passed between the components of the client. Together with the communication a certain amount of computation has to be performed in order to decide where the vehicle is and necessary course corrections. The amount of computation that has to be done varies only slightly from one loop to the next, so what remains that can affect the time it takes to complete the loop is the time it takes for the components to communicate with each other. In order to determine how good the communication component was we decided to gather the following three datapoints:

- The time it takes for a component to send a message.

- The time it takes for the communication component to deliver the message (message transit time).
- The time it takes to complete a navigation loop in our architecture model.

Aside from the pure performance questions there were two additional questions, i.e., questions that we did not intend to focus the prototype on but that we would keep an eye out for in order to get a feel for how the communication component handled them.

- Is there a large difference in performance between Linux and Windows 2000? This was a concern raised by some of the engineers developing the system.
- How easy is it to port the communication component from Windows 2000 to Linux?

So now we have defined the goal of our evaluation and we have defined the data that we will need in order to perform the evaluation.

4.2. Implement an evaluation support framework

Based on the defined goal of the evaluation we created an evaluation support framework that would handle the gathering of data as well as separate the communication component from the architecture model. The conceptual model for the evaluation support framework that we constructed consisted of four main concepts: worker, log, framework, and communication component.

- A worker is someone that performs work such as calculations based on incoming data. A worker can both produce and consume messages. Instances of the worker are used to create the architecture model.
- The log is used to store information regarding the time it takes to perform tasks in the system. The framework uses the log to store information regarding sent and received messages.
- The framework provides a small API for the workers. It is mainly responsible for sending relevant events to the log but it also provides methods for generating unique message id:s as well as sending and receiving messages etc.
- The communication component is some kind of communication method that we are interested in evaluating. This is the exchangeable part of the framework

The four concepts were realised as a small number of classes and interfaces, as shown in Figure 2. The evaluation support framework provided an abstract worker class that contained basic functionality for sending and receiving messages. This class also generated log entries for each event that happened (such as the sending or receiving of a message). The class was also responsible for shutting down the testing once a predetermined time had elapsed, during the shutdown the log was flushed to disk. When creating a worker the user extended the abstract class and through that got all the messaging an logging functionality ready to use. Only some initialization was left to be done.

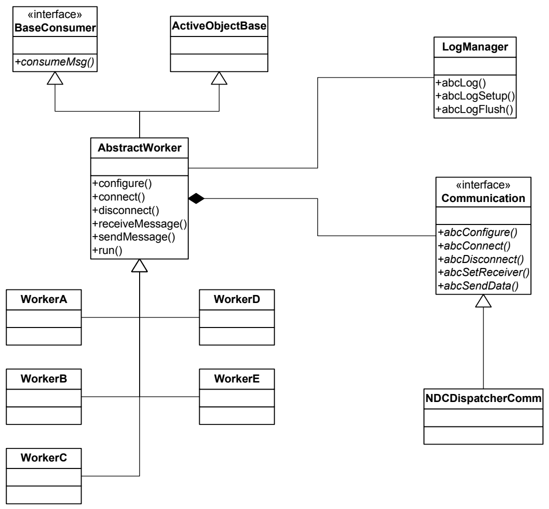

Figure 2: A class diagram of the simulation framework.

The log was realised in a LogManager class that stored log entries together with a timestamp for each entry. All log entries were stored in memory during the execution of the model and written to disk first after the execution had ended. This construction was chosen as it ensured that there, during execution, was no disk activity other than what was initiated by the OS, workers, or communication component.

The communication is represented by an interface in the framework. The interface only provided basic methods for starting up, configuring, shutting down, sending, receiving, connecting and disconnecting. If a new communication component is to be tested, a wrapper class is written that implements the communication interface and is able to translate the calls from the framework to the communication component.

4.3. Integrate architectural components

The communication component that we wanted to evaluate was integrated with the evaluation support framework. The component provided asynchronous communication based on publisher-subscriber, meaning that all components that are interested in some type of message subscribes to it using the communication component. When a message is sent to the communication component it is published to all the components that have subscribed to that type of message.

The communication interface for the framework was implemented by a wrapper class that passed on messages to be sent and received. It also handled the translation of telegram types from the support framework to the communication component.

4.4. Implement architecture model

With the framework in place and ready to use we went on to create the architecture model. It was built based on the architecture of the navigation system and focused on modeling the navigation loop. During the loop, several components interact with each other and perform computations as a response to different messages, as shown in Figure 3.

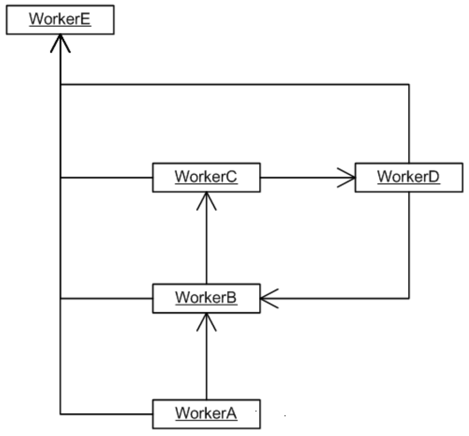

Figure 3: The model of the different workers that interact during the simulation.

Workers B, C, and D are the most critical components of the navigation loop. WorkerB initiates the loop when it receives a message from WorkerA which sends a message periodically every 50 ms. When workerB receives a message it works for 1 ms and then sends a message with its results. WorkerC then receives the message from workerB and it proceeds to work for 5 ms before sending its message. This message is in turn received by workerD that also works for 5 ms before sending a new message. The message from workerD is received by workerB which notes that it has been received but does nothing more. WorkerE has subscribed to all the message types that exist in the model and thus receives all messages that are sent between the other workers.

In order to keep the model simple and easy to understand we simplified the interactions between the components so that each component only published one type of message.

Since we wanted to evaluate the performance of the communication component without affecting the system too much during execution we logged only time stamps together with a small identifier in runtime. This kept the time it took to create a log entry to a minimum. Log entries were created every time a worker entered or returned from the communication component and also when a message was actually sent. When sending a message we logged a time stamp together with the whole message.

4.5. Execute prototype

The prototype was executed on three different hardware platforms, these were all Intel Pentium based platforms at the speeds of 133 MHz, 700 MHz, and 1200 MHz. The operating systems that were used were Linux, Windows 2000, and Windows XP. The operating system that the client is targeted for is Linux but all development is performed on Windows 2000 and test runs of the system are performed on the development platform. Therefore we wanted to run our prototype on that operating system as well. It also helped us to determine how portable the communication component was.

The architecture prototype was executed during 20 seconds on each platform and the logs from all runs were gathered and stored for further analysis.

Each execution of the prototype resulted in roughly half a megabyte of logged data. The data was stored in five separate log files (one for each worker) and the individual size of the log files varied between 50 and 150 KB. Each log entry was between 50 and 70 bytes and we gathered about 7000 log entries.

Based on the timestamps and identifiers from the logs we were able to extract information regarding two things.

- The time it took from that a worker called the send message function until the method returned. This measure is important as it is the penalty that the worker has to pay for sending a message.
- How long time any given message spent "in transit," i.e., how long time it took from that a worker sent a message until it was received by the recipient or recipients.

Measurements were also made in order to see how much overhead that was added to the communication by the evaluation support framework. We found that the framework added a delay of between 6 to 15 percent to the time it took to send a message. This results in 0.1 to 0.3 ms on the average send of a message on the Pentium 133Mhz based machine with Linux as OS (Table 1).

4.6. Analyse logs

We implemented a small program that parsed the generated logs and extracted the information that we wanted from the data files. Based on the time stamps and identifiers from the logs we were able to extract information that we had defined as necessary for evaluating the communication component, namely:

- The time it takes for a component to send a message (Table 1).
- The time it takes for the communication component to deliver the message (Table 2).
- The time it takes to complete a navigation loop in our architecture model (Table 3).

In all the tables, L stands for Linux, W2K stands for Windows 2000, WXP stands for Windows XP. The number is the speed of the Pentium processor, N stands for NFS mounted system and F stands for a system with a flash memory disk. Windows 2000 and Windows XP test used hard drives.

Table 1: The table shows the time it took for a client to execute the method for sending a message. Values are in microseconds.

OS/ HW	L133 N	L133 F	L700 N	L700 F	W2K 700	WXP 1200
Min	1562	1571	88	90	88	38
Med	1708	1716	97	97	97	67
Max	2093	2601	645	280	645	163

Table 2: The table shows how long time a message spent in transit from sender to receiver. Values are in microseconds.

OS/ HW	L133 N	L133 F	L700 N	L700 F	W2K 700	WXP 1200
Min	921	922	55	55	55	18
Med	3095	3094	1174	1173	1174	1157
Max	9241	9228	5076	5061	5076	5063

Table 3: The table shows the time it took for the prototype to complete a navigation loop. Values are in microseconds.

OS/ HW	L133 N	L133 F	L700 N	L700 F	W2K 700	WXP 1200
Min	22765	22712	12158	12158	12096	12067
Med	22840	22834	12161	12159	12130	12100
Max	23128	23128	12165	12163	12245	12190

In Table 3 we can see that the average time that it takes to complete a navigation loop on the L133 platforms is 22,8 ms. This figure can be broken down into two parts: work time and message transit time. During the loop workerB and workerC has to perform 10 ms of work and besides this workerE has to perform 3 times 1 ms of work resulting in 13 ms total work time. The average message transit time of about 3,1 ms per message adds up to an average of 9,3 ms for communication during the loop. The figures add up to 22,5 ms for a navigation loop where only three messages are delivered. The fact that we spend about 40% of the time on communicating is taken as an indication that the communication component is unsuitable for the Pentium 133 MHz based system.

4.7. Predict quality attributes

Based on the information that were extracted from the logs, we concluded that the 133 Mhz Pentium based platform probably would be unable to fulfill the performance requirements for the systems. The time it took to dispatch a message was far to great to be able to handle the amounts of messages that the real system would generate.

Regarding the additional questions about the difference between operating systems and portability, we were able to draw the following conclusions:

- We found that the expected difference in performance between Windows 2000 and Linux didn't exist. The two operating systems performed equally well in the evaluation with less than 1% performance difference between the two.
- We showed that it would be easy to port the component from one platform to another, and from one compiler to another. All that was necessary for the port to build on Linux was that the necessary makefiles were written. The high portability of the system was attributed to the use of the ACE framework together with following the ANSI C++ standard.

4.8. Reiterate if necessary

After a first iteration of all the steps in the evaluation method, some questions were raised regarding how the evaluation support framework handled the logging of data. The first version of the support framework flushed the communication logs to disk every 100 log entry. This could cause the OS to preempt the architecture prototype in order to complete the write operation. This in turn lead to spikes in the time it took for a message to be sent and received. In order to remove this problem the framework was changed so that the logs were stored in memory during the execution of the prototype and flushed to disk just before the prototype exited.

5. Results from the case study

The evaluation we performed in the case study resulted in that some fears regarding the performance of the new client for the AGV system were confirmed and that the developers took steps to investigate possible solutions to the problem. The evaluation also successfully answered the additional questions that were posed in the beginning of the evaluation.

When the implementation of the new system had become stable we made measurements on the new system in order to validate the results from the prototype. The data was gathered at the same points in the code as the evaluation support framework did and the measurements showed that the prototype produced the same message delivery times as the real system.

6. Analysis of the evaluation method

We feel confident that the prototype based evaluation approach is useful for assessing the performance characteristics of an architecture component and also for evaluating the performance of architecture models derived from proposed software architectures. New architecture models are easily implemented using the evaluation support framework and the amount of reuse, both in code and analysis tools makes the creation of a support framework and analysis tools worth the effort.

The support framework separates the component that we are evaluating from the architecture model, making it possible to compare alternative components in a consistent way as the data for the evaluation is gathered in the same way independently of the component.

A concern that can be raised against the use of an evaluation support framework is that since a message has to go through the framework classes before it reaches the component that we are evaluating there is an added delay. In our case study we found that the delay between that the worker sent the message and that the message was actually sent by the interaction component was quite small. The framework added between 0,1 to 0,3 ms to the time it took to send a message.

7. Future work

We plan to continue to use the test framework and the test applications and try to evaluate other quality attributes such as scalability and maintainability.

The test-framework and applications will be used to perform follow-up evaluations at Danaher Motion Särö AB in order to see how the communication component develops, and ultimately to compare how well the original estimations match the performance of the final system.

8. Conclusion

In this paper we have described the prototype based architecture evaluation approach and the steps that it consists of. The approach is based on the simulation based evaluation approach but adds mainly the construction of an evaluation support framework and a clearer focus on iteration during the evaluation. The use of the evaluation support framework simplifies the implementation of alternative architecture models, makes consistent gathering of data simpler, and makes it possible to evaluate alternative implementations of an architecture component.

The evaluation approach has been used to evaluate the performance characteristics of a communication component in an AGV system architecture. The evaluation resulted in that a performance problem was identified and that two additional questions were evaluated as well (Portability and performance differences between Windows 2000 and Linux). Results from the case study were also used to validate the evaluation approach once the new system was stable, and showed that it produced accurate results.

The case study illustrates the steps within the evaluation process and can be seen as a guide for performing a prototype based evaluation.

Acknowledgments

This work was partly funded by The Knowledge Foundation in Sweden under a research grant for the project "Blekinge - Engineering Software Qualities (BESQ)" (http://www.ipd.bth.se/besq).

We would like to thank Danaher Motion Särö AB [6] for providing us with a case for our case study and many interesting discussions and ideas.

References

[1] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, Addison-Wesley, 1998.

[2] Bosch J.: *Design & Use of Software Architectures*, Pearson Education Limited, ISBN 0-201-67494-7.

[3] C. Smith and L. Williams, “Performance Solutions - A Practical Guide to Creating Responsive, Scalable Software,” Addison-Wesley, 2001.

[4] D.E. Perry and A.L.Wolf, ‘Foundations for the Study of Software Architecture,’ *Software Engineering Notes*, 17(4):40-52, October 1992.

[5] M. Shaw and D. Garlan, *Software Architecture - Perspectives on an Emerging Discipline*, Prentice Hall, 1996.

[6] Danaher Motion Särö AB, http://www.danahermotion.se