Prototype-based Software Architecture Evaluation — Component Quality Attribute Evaluation

Frans Mårtensson, Håkan Grahn, and Michael Mattsson

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

ABSTRACT

The architecture of a software system is crucial since it often affects and limits the quality attributes of the system, e.g., performance and maintainability. In this paper we evaluate three communication components suggested for use in a software architecture using prototype-based evaluation. We evaluate the performance quantitatively, and also present qualitative results on portability and maintainability. Components for both intra- and inter-process communication are evaluated and we find that it might be possible to use one third-party component for both intra- and inter-process communication, thus replacing two inhouse developed components.

KEYWORDS: Software architecture, performance evaluation, component evaluation.

1. Introduction

The size and complexity of software systems are constantly increasing. It has been identified that the quality properties of software systems, e.g., performance and maintenance, often are constrained by their software architecture [1]. The software architecture describes the parts that make up a software system, their responsibilities, and how they interact with each other. The software architecture is created early in the development of a software system and has to be kept alive throughout the system life cycle. Part of the process of creating a software architecture is the decision of possible use of existing software components in the system.

Before committing to the use of a particular software architecture and set of components, it is important to make sure that it will be able to handle all the requirements that are put upon it. Bad architecture design decisions can result in a system with undesired characteristics, e.g., low performance and/or low maintainability. Therefore, evaluating the quality properties of a proposed software architecture is very important.

Several approaches to architecture evaluation can be identified [2]. In this study we use an prototype-based evaluation approach from the simulation family of evaluation methods that relies on the construction of an executable prototype of the architecture. Small test implementations of candidate technologies and architectures can easily be implemented and several alternatives can be compared.

In this paper we evaluate the quality properties of three different communication components in a distributed system. We use three prototypes built using the same prototype framework to measure the performance of each component, both intra-process as well as inter-processes communication are tested. The communicating processes reside both on the same computer and on two different computers connected with a network. We also discuss qualitative data for the portability and maintenance aspects of the components.

The evaluation is performed in cooperation with Danaher Motion Särö [4] and the usage scenarios that are used during the evaluations have been developed in cooperation with them.

The paper is organized as follows. Section 2 presents some background to software architecture, architecture evaluation and automated guided vehicles. In Section 3 we introduce the components and the quality attributes that we will evaluate. We present the results from the evaluation in Section 4. Finally, in Section 5 we conclude our study.

2. Background

In this section, we give some background about software architectures, how to evaluate them, different quality attributes, and the application domain, i.e., automated vehicle systems.

2.1. Software Architecture

Software systems are developed based on a requirement specification. The requirements can be categorized into *functional* requirements and *non-functional* requirements, also called *quality requirements*. Functional requirements are often easiest to test (the software either has the required functionality or not) and the non-func-

tional requirements are harder to test (quality is hard to define and quantify).

In the recent years, the domain of software architecture [1, 2, 8, 9] 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.

Software architecture is defined in [1] as follows:

"The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them."

Software 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. However, the architecture of a software system is fundamental to its structure and cannot easily be changed without affecting virtually all components and, consequently, considerable effort.

2.2. Architecture Evaluation Methodology

In order to make sure that a software architecture fulfills its quality requirements, it has to be evaluated. Four main approaches to architecture evaluation can be identified, i.e., scenarios, simulation, mathematical modelling, and experience-based reasoning [2]. In this paper we use a prototype-based architecture evaluation method which is part of the simulation-based approach and relies on the construction of an executable prototype of the architecture [2, 5, 10]. It also lets the developer compare all components in a fair way, all components get the same input from a simplified architecture model. An overview of the parts that go into a prototype is shown in Figure 1. A strength of this evaluation approach is that it is possible to make accurate measurements on the intended target platform for the system early on in the development cycle.

The prototype-based evaluation is performed in seven steps plus reiteration. We will describe the steps shortly in the following paragraphs.

Define the evaluation goal. 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 an evaluation support framework. The evaluation support framework's main task is to gather data that is relevant to fulfilling the evaluation goal. Depending on the goal of the evaluation, the support framework has to be designed accordingly, but the main task of the support framework is to simplify the gathering of data. The support framework can also be used to provide common functions such as base and utility classes for the architecture models.

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. The easiest way of achieving this is to create a proxy object that translates calls between the framework and the component.

Implement architecture model. Implement a model of the architecture with the help of the evaluation support framework. The model should approximate the behavior of the completed system as far as necessary. The model together with the evaluation support framework and the component that is evaluated is compiled to an executable prototype.

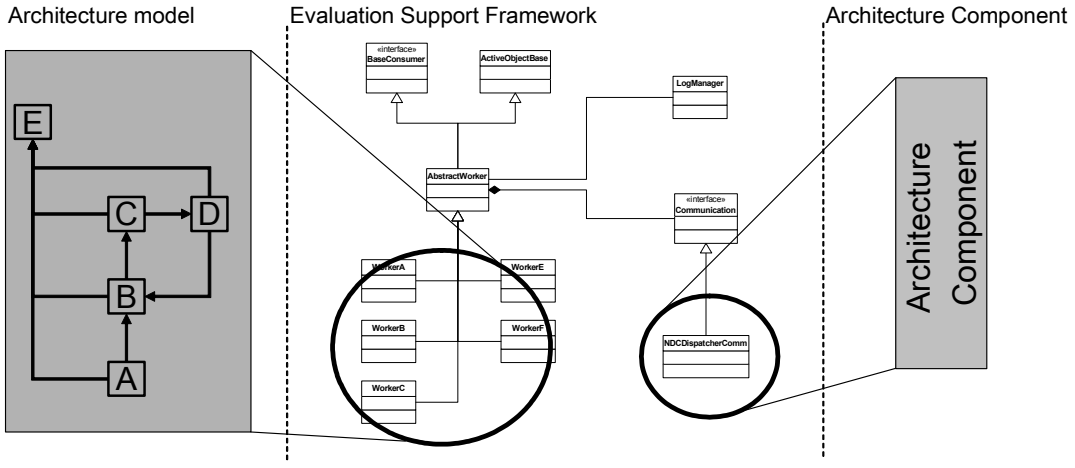

Figure 1: The prototype consists of three main parts: the architecture model, the evaluation support framework, and the architecture components.

Execute prototype. Execute the prototype and gather the data for analysis in the next step. Try to 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.

Reiteration. 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 should 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. It is also 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.

2.3. AGV Systems

As an industrial case we use an Automated Guided Vehicle system (AGV system). AGV systems are used in industry mainly for supply and materials handling, e.g., moving raw materials, and finished products to and from production machines.

Central to an AGV system is the ability to automatically drive a vehicle along a predefined path, the path is typically stored in a path database in a central server and distributed to the vehicles in the system when they are started. The central server is responsible for many things in the system, it keeps track of the vehicles positions and uses the information for routing and guiding the vehicles from one point in the map to another. It also manages collision avoidance so that vehicles do not run into each other by accident and it detects and resolves deadlocks when several vehicles want to pass the same part of the path at the same time. The central server is also responsible for the handling of orders from operators. When an order is submitted to the system, e.g., “go to location A and load cargo”, the server selects the closest free vehicle and begins to guide it towards the pickup point.

In order for the central server to be able to perform its functions, it has to know the exact location of all the vehicles under its control on the premise. Therefore every vehicle sends its location to the server several times every second. The vehicles can use one or several methods to keep track of its location. The three most common methods are induction wires, magnetic spots, and laser range finders.

The simplest way is the use of induction wires that are placed in the floor of the premises. The vehicles are then able to follow the electric field that the wire emits and from the modulation of the field determine where it is.

A second method of navigation is to place small magnetic spots at known locations along the track that the truck is to follow, the truck can then predict where it is based on a combination of dead reckoning and anticipation of coming magnetic spots.

A third alternative is the use of a laser located on the vehicle, that measures distances and angles from the vehicle to a set of reflectors that has been placed at known locations throughout the premises. The control system in the vehicle is then able to calculate its position in a room based on the data returned from the laser.

Regardless of the way that a vehicle acquires the information of where it is, it must be able to communicate its location to the central control computer. Depending on the available infrastructure and environment in the premises of the system, it can for example use radio modems or a wireless LAN.

The software in the vehicle can be roughly divided into three main components that continuously interact in order to control the vehicle. These components require communication both within processes and between processes located on different computers. We will perform an evaluation of the communication components used in an existing AGV system and compare them to an alternative COTS component for communication that is considered for a new version of the AGV system.

3. Component Quality Attribute Evaluation

In this section we describe the components that we evaluate, as well as the evaluation methods used. The goal is to assess three quality attributes e.g. performance, portability and maintainability for each component. The prototypes simulate the software that is controlling the vehicles in the AGV system. The central server is not part of the simulation.

3.1. Evaluated Communication Components

The components we evaluate are all communication components. They all distribute events or messages between threads within a process and/or between different processes over a network connection. Two of the components are developed by the company we are working with. The third component is an open source implementation of the CORBA standard.

NDC Dispatcher. The first component is an implementation of the dispatcher pattern which provides publisher-subscriber functionality and adds a layer of indirection between the sender and receivers of messages. It is used for communication between threads within one process

and can not pass messages between processes. The dispatcher is implemented with active objects using one thread for dispatching messages and managing subscriptions. It is able to handle distribution of messages from one sender to many receivers. The implementation uses the ACE framework for portability between operating systems. The component is developed by the company and is implemented in C++.

Network Communication Channel. Network Communication Channel (NCC) is a component is developed by the company as well. It is designed to provide point to point communication between processes over a network. It only provides one to one communication and has no facilities for managing subscriptions to events or message types. NCC can provide communication with legacy protocols from previous versions of the control system and can also provide communication over serial ports. The component is implemented in C.

TAO Real-time Event Channel. The third component, The ACE Orb Real-time Event Channel (TAO RTEC) [7], can be used for communication between both threads within a process, and between processes both on the same computer and over a network. It provides communication from one to many through the publisher subscriber pattern. The event channel is part of the TAO CORBA implementation and is open source. This component can be seen as a commercial of-the-shelf (COTS) component to the system. We use TAO Real-time Event Channel to distribute messages in the same way that the Dispatcher does.

3.2. Software Quality Attributes to Evaluate

In our study we are interested in several quality attributes. The first is performance because we are interested in comparing how fast messages can be delivered by the three components. We assess the performance at the system level and look at the performance of the communication subsystem as a whole.

The second attribute is portability, i.e., how much effort is needed in order to move a component from one environment/platform to another. This attribute is interesting as the system is developed and to some extent tested on computers running Windows, but the target platform is based on Linux.

The third attribute is maintainability which was selected since the system will continue to be developed and maintained under a long period. The selected communication component will be an integral part of the system, and must therefore be easy to maintain.

Performance. We define performance as the time it takes for a communication component to transfer a message from one thread or process to another. In order to measure this we created one prototype for each communication component.

Four prototypes were constructed using a framework that separated the communication components from the model of the interaction. By doing so we were able to use the same interaction model for the prototypes and minimize the risk of the communication components being treated unequally in their test scenarios. Most of the framework was reused from a previous study [5] and only minor modifications were introduced in it for this study. Most notably we added functionality for measuring the difference in time for computers that were connected via a network. This information was used to both adjust the timestamps in the logs when prototypes were running on separate computers, and to synchronize the start time for when the prototypes should start their models.

We created two different models, one for communication between threads in a process and one for communication between processes on separate computers that communicated via a network. The NDC Dispatcher was tested with the model for communication between threads in a process and NCC was tested with the model for communication over the network. The TAO RTEC was tested with both models since it can handle both cases.

The prototypes were executed three times on a test platform that was as similar to the target environment as we could make it. The test setup consisted of two computers running the Linux 2.4 kernel. both computers had a 233Mhz Pentium 2 processor and 128 MB RAM. Both computers were connected to a dedicated 10Mbps network for the tests that required network communication.

Portability. We define portability as the effort needed to move the prototypes and communication components from the Windows XP based platform to the Linux 2.4 based platform. This is a simple way of assessing the attribute but it verifies that the prototypes actually works on the different platforms and it gives us some experience from making the port. Based on this experience we can make a qualitative comparison of the three components.

Maintainability. We reason around this attribute using qualitative discussions and our experiences from development of the prototypes. We have also looked at the size of the components using the lines of code count as an indication of complexity.

4. Evaluation Results

During the evaluation, the largest effort was devoted to implementing the three prototypes and running the performance benchmarks. The data from the performance benchmarks gave us quantitative performance figures which together with the experience from the implementations were used to assess the portability and maintainability of the components.

4.1. Performance Results

After implementing the prototypes and performing the test runs, the gathered logs were processed by an analysis tool that merged the log entries, compensated for the differences in time on the different machines and calculated the time it took to transfer each message

4.1.1. Intra Process Communication

The intra process results in Table 1 shows that the average time it takes for the Dispatcher to deliver a message is 0,3 milliseconds. The same value for the TAO RTEC is 0,6 milliseconds. The extra time that it takes for TAO RTEC is due to the differences in size between it and the Dispatcher. The TAO RTEC makes use of a CORBA ORB for dispatching the events between the threads in the prototype, this makes the TAO RTEC very flexible but it impacts its performance when both publisher and subscriber are threads within the same process; the overhead in a longer code path for each message becomes the limiting factor. The Dispatcher on the other hand is considerably smaller in its implementation than the TAO RTEC, resulting in a shorter code path and faster message delivery.

Table 1: Communication time between components when thay are on the same computer.

	Dispatcher	**TAO RTEC**	**NCC**
Intra process	0,3 ms	0,6 ms	
Inter process		2 ms	0,8 ms

During the test runs of the Dispatcher and TAO RTEC based prototypes we saw that the time it took to deliver a message was not the same for all messages. Figure 2 and Figure 3 show a moving average of the measured times in order to illustrate the difference in behavior between the components. In both the Dispatcher and TAO RTEC prototypes this time depends on how many subscribers that there are to the message, and the order that the subscribers subscribed to a particular message. We also saw that when using TAO RTEC there is a large variation from message to message. It is not possible to guarantee that the time it takes to deliver a message will be constant when using either the Dispatcher nor the TAO RTEC, but the Dispatchers behavior is more predictable.

4.1.2. Inter Process Communication

The inter process communication in Table 2 shows that the TAO RTEC takes on average 2 milliseconds to deliver a message from one computer to another in our test environment. The NCC component takes on average 1 millisecond. The extra time needed for TAO RTEC to deliver a message is also a result of the longer code path involved due to the use of CORBA. The gain of having this component is added flexibility in how messages can be distributed between subscribers on different computers. In comparison, the NCC component is only able to pass messages from one point to another, making it less complex in its implementation.

Figure 2: Dispatcher delivery time.

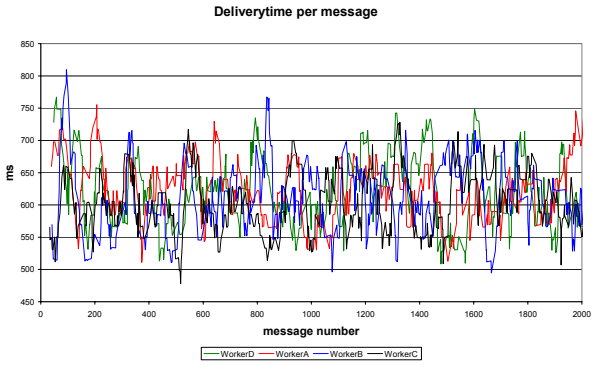

Figure 3: TAO RTEC delivery time.

Table 2: Communication time between components when thay are on different computers

	TAO RTEC	**NCC**
Inter process	2 ms	1 ms

In Table 3 we see the amount of data that is actually transmitted (and in how many TCP/IP packages) over the network by prototypes using TAO RTEC and NCC respectively. In the architecture model, both prototypes perform the same work and sends the same number of messages over the network. In the table we see that both components send about the same number of TCP/IP packages (TAO RTEC sends 37 more than NCC), the difference is located to the initialization of the prototypes were a number of packages are sent during ORB initialization, name resolution and subscriptions to the event channel etc. When we look at the amount of data sent in the packages we see that TAO Real-time Event Channel sends about 55% more data than NCC does. This indicates that NCC has less overhead per message than TAO Real-time Event Channel does.

Both components do however add considerably to the amount of data that is generated by the model which generated 6 kb of data in 300 messages.

Table 3: Network traffic generated by TAO RTEC and NCC.

	TAO RTEC	**NCC**
TCP/IP packages	800 packages	763 packages
Data over network	137 kb	88 kb

4.2. Portability Results

Based on our experiences from building the prototypes we found that moving the prototypes from the Windows based platform to the Linux based platform was generally not a problem and did not take very long time (less than a day per prototype), most of the time was spent on writing new make files and not on changing the code for the prototypes.

Both the Dispatcher and TAO Real-time Event Channel are based on the ADAPTIVE Communications Environment (ACE) [7] that provides a programming API that has been designed to be ported to many platforms. Once ACE was built on the Linux platform it was easy to build the prototypes that used it.

NCC was originally written for the Win32 API and uses a number of portability libraries built to emulate the necessary Win32 APIs on platforms other than windows. Building the prototype using NCC was not more complicated than those using the Dispatcher or TAO RTEC.

4.3. Maintainability Results

The Dispatcher is quite small (700 loc) and therefor quite easy to get an overview of. This component is probably the easiest for a maintainer to understand.

TAO is in comparison very large (>400 kloc). Although the parts that are used for the real-time communications channel are smaller it is still difficult to get an overview of the source code. The question of maintainability is relevant only if one version of TAO is selected for continued use. If newer versions of TAO are used as they are released then the maintenance is continuously done by the developer community around TAO. There is however the possibility that APIs in TAO are changed during development, breaking applications. But since the application developers are with the company, this problem should be easier to deal with than defects in TAO itself.

The NCC is larger than the dispatcher but not as large as TAO. Unfortunately we have not been able to review the code and structure of NCC, which makes it difficult for us to make a statement about the maintainability of this component.

5. Conclusions and Future Work

In this paper we have used a prototype-based evaluation methods for assessing three different communication components. We have shown that it is possible to compare the three different communication components in a fair way using a common framework for building the prototypes and analyzing the resulting data. The components were one COTS component, The ACE Orb Real-Time Event Channel (TAO RTEC), and two inhouse developed components, Dispatcher and NCC.

We evaluate three quality attributes: performance, portability, and maintainability. The performance is evaluated quantitatively, while portability and maintainability are evaluated qualitatively.

The performance measurements show that TAO RTEC is slower than the Dispatcher in communication between threads within a process, and also that it is slower than NCC in communication between processes.

All three components fulfill the portability requirement in this study. We had no problems moving the prototypes from a Windows based- to a Linux based platform.

The maintainability of the components weight in the favour of the Dispatcher and NCC, both components have been developed within the company and the knowledge of how they are constructed is documented. TAO RTEC is the largest of the three components and the knowledge of how it is constructed is not within the company. Therefore we think that TAO RTEC is less maintainable for the company. On the other hand, the company can take advantage of future development of TAO RTEC with no effort as long as the APIs are the same.

Future work is to validate the accuracy of the prototypes, this will be possible if the system is constructed using the components we have evaluated. We would also like to evaluate the usefulness of the prototypes from the developers point of view, if the prototypes are a good means for spreading knowledge between developers. We will use questionnaires to assess the knowledge levels of the developers before and after they are presented with the prototypes. The goal will be to try to assess if prototypes are a good method for spreading knowledge of a new technologies and methods within a company. The reason is that if a prototype has been built in order to evaluate a new technology then the effort has already been spent, so if there anything to gain from sharing the prototype with others then that will be a good thing.

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.bth.se/besq.

We would like to thank Danaher Motion Särö AB [4] 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] J. Bosch: *"Design & Use of Software Architectures,"* Pearson Education Limited, ISBN 0-201-67494-7.

[3] L. Dobrica, E. Niemela, "A Survey On Architecture Analysis Methods," *IEEE Transactions on Software Engineering,* 28(7):638 - 653, July 2002.

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

[5] F. Mårtensson, H. Grahn, and M. Mattsson, "An Approach for Performance Evaluation of Software Architectures using Prototyping," *Proc. IASTED Int'l Conference on Software Engineering and Applications (SEA 2003)*, pp. 605-612, Nov. 2003.

[6] Object Management Group, *"CORBA™/IIOP™ Specification, 3.0,"* Mar. 2004, available at www.omg.org.

[7] D. Schmidt et al, "The ACE ORB", available at http://www.cs.wustl.edu/~schmidt/TAO.html last checked Sept. 2004.

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

[9] M. Shaw and D. Garlan, *"Software Architecture - Perspectives on an Emerging Discipline,"* Prentice Hall, ISBN 0-13-182957-2.

[10] C. Smith and L. Williams, *"Performance Solutions - A Practical Guide to Creating Responsive, Scalable Software,"* Addison-Wesley, 2001.