Abstract—This paper studies the problem of predicting the performance of a component-based application based on the performance of its components. The paper is based on a literature study. It presents a few methods of performance prediction, and discusses whether they might be feasible in an industrial setting.

Index Terms—performance prediction, component-based software, predictable assembly

1. INTRODUCTION

There is a trend in software engineering that more and more software systems are built from components. By reusing existing components, their development cost can be amortized over many systems.

The components can be developed by the organization developing the system (in-house development), or they can be procured from companies selling ready-made components.

In the former case, the organization might develop a range of similar systems, and hope to achieve cost-reductions by reusing already developed parts.

In the latter case, there is usually a broad need for certain services, which allows a company to develop components providing those services and sell them to other companies to use as parts of their systems.

When planning the system, it is often possible to predict the cost of the system as a function of the cost of the components involved.

In the cases where the components can be procured from an external source, or where the component has already been developed in-house, the cost of the components is well known. In the cases where the components need to be developed in-house, the cost of the components is a little harder to predict, but still possible.

Likewise the functionality of the system is usually designed based on the functionality of the components (or to be more precise, the components are chosen based on the functionality of the system). The components provide certain services, and require others, and by combining components, the system functionality is realized.

Often a system has other requirements than the purely functional. These non-functional quality attributes can sometimes be quite important for the overall quality of the system. For example in safety-critical systems, the reliability of the system can be of utmost importance, and in some systems the performance of the system can be quite important because of resource constraints.

It would therefore be very important to be able to predict also the non-functional quality attributes of the system based on the attributes of the components used.

1.1 Research problem

This paper will study whether it is possible to predict certain non-functional quality attributes of a component-based system based on the attributes of its components.

The paper will focus on the performance quality attribute. Other attributes will not be studied.

The paper will study what different methods there are for predicting the performance of a composed system, and find out how/if they have been validated. The paper will also discuss the feasibility of using these methods in an industrial setting.

The paper is based on a literature study, which studies different methods for predicting the performance attribute of a composed system. The performance attribute was chosen because it is one of the attributes that has been studied the most.

1.2 Structure of the paper

In the next section, the paper will take a closer look at the performance attribute.

In section 3, the paper will explain what is meant by "predictable assembly", and present some research done in the area. It will present some of the underlying philosophy behind the concept.

In section 4, the paper will present some methods for predicting the performance attribute, and present the validation (if any) that has been done on the methods.

In section 5, the author will discuss the results presented in the previous sections. The author will evaluate whether the results show that the performance of a composed system can be predicted, and whether the methods could provide benefits in an industrial setting.

In the last section, the paper will summarize the discussion, and present the conclusions made.

2. THE PERFORMANCE QUALITY ATTRIBUTE

When it comes to predicting the quality attributes of a composed system, some quality attributes are easier to predict than others. In his PhD thesis (Larsson, 2004), Magnus Larsson presents a taxonomy of quality attributes. He
categorizes the quality attributes based on their compositability. He presents five different categories of properties.

2.1 Directly composable properties

According to his definition, a directly composable property is one in which the property of the assembly is a function only of the same property of the components. He mentions as an example the static memory footprint, which can be calculated as the sum of the memory footprints of the components. These properties are the easiest to predict.

2.2 Architecture-related properties

The definition of an architecture-related property is that it is one where the property of the assembly is a function both of some property of the components, and of the architecture of the system.

This means that in order to be able to predict the property of the assembly, knowledge is needed about how the components are connected to form the assembly. Larsson mentions reliability as an example. Another example that is mentioned is performance.

2.3 Derived properties

The definition of a derived property is that the property of the assembly is a function of several different properties of the components. As an example of such a property, Larsson mentions the end-to-end deadline of a real-time system. This property depends on both the worst-case execution time and the execution period.

He also mentions that there are properties which are not visible on the component level, but which are interesting on the assembly (or system) level. He does not mention any examples of such a property, but suggests that the hardest thing about these properties is to identify which component properties they depend on.

<table>
<thead>
<tr>
<th>Property</th>
<th>Directly composable</th>
<th>Architecture-related</th>
<th>Derived</th>
<th>Usage-dependent</th>
<th>System environment context</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>**</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Footprint</td>
<td>**</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Responsiveness</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scalability</td>
<td></td>
<td>**</td>
<td></td>
<td>**</td>
<td></td>
</tr>
<tr>
<td>Schedulability</td>
<td></td>
<td>**</td>
<td>**</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Timeliness</td>
<td></td>
<td>**</td>
<td>**</td>
<td>**</td>
<td></td>
</tr>
<tr>
<td>CPU utilization</td>
<td></td>
<td>**</td>
<td>**</td>
<td>**</td>
<td></td>
</tr>
<tr>
<td>Latency</td>
<td></td>
<td>**</td>
<td>**</td>
<td>**</td>
<td></td>
</tr>
<tr>
<td>Transaction throughput</td>
<td></td>
<td>**</td>
<td>**</td>
<td>**</td>
<td></td>
</tr>
<tr>
<td>Concurrency</td>
<td></td>
<td></td>
<td>**</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Efficiency</td>
<td></td>
<td>**</td>
<td>**</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2.4 Usage-dependent properties

The definition of a usage-dependent property is that it not only depends on the properties of the components but also on the usage profile. How the system is used has an impact on these properties.

According to Larsson, these kinds of properties are problematic to reason about, since it is difficult to determine how to transfer the usage profile of the assembly to the individual components.

2.5 System environment context properties

The definition of a system environment context property is that it is determined not only by other properties, or the usage profile, but also by the state of the system environment.

As an example of such a property, Larsson mentions safety. He explains that under different circumstances the system might exhibit different degrees of safety although the usage profile is the same. This means that these properties are the ones that are hardest to predict.

Larsson suggests that the analysis approach for these properties should be the opposite of composition; rather these properties are decomposed to get the required component properties.

In his dissertation (Larsson, 2004), Larsson also shows a classification of the different quality attributes according to the categories presented above. Since this paper will focus on the performance attribute, only the classification of performance attributes will be included in this paper. Table I shows the classification of the performance properties.

According to his classification, most of the performance properties fall into one of the first three categories.

Larsson validated the classification by surveying twelve researchers. The researchers answered a questionnaire were they could indicate to which category they thought a quality attribute belonged, and indicate how confident they were in their assessment. The researchers did not completely agree
with Larsson’s classification. Larsson suggests that it might be because the quality attributes were insufficiently described in the questionnaire, leading to differences of opinion about what the attributes meant.

3. PREDICTABLE ASSEMBLY

In a white paper (Crnkovic et al., 2002) predictable assembly is defined as "the activity of predicting properties of assemblies of components prior to actually acquiring the components". The paper offers a model research outline for projects studying predictable assembly, and defines some vocabulary related to the problem. The paper was used as a guideline to judge whether research papers were suitable for the Fifth ICSE Workshop on Component-Based Software Engineering, which was focused on predictable assembly of component-based systems.

A technical report from the Software Engineering Institute (Wallnau, 2003) explains that "an assembly of software components is predictable if its runtime behavior can be predicted from the properties of its components and their patterns of interaction." It further defines a component as certifiable if its properties "can be verified by independent third parties". The report then goes on to describe a research project performed at the SEI, which studies the "predictable assembly of certifiable components".

According to a paper by Stafford and Wallnau (Stafford and Wallnau, 2001a) the quality of a system depends on how predictably its features interact. Many of the quality attributes of a system depend on the interaction between the components that the system is composed of. Stafford and Wallnau claim that a failure to achieve a certain quality attribute can sometimes be due to the unanticipated interaction of the system's features.

In their paper, Stafford and Wallnau mention compositional reasoning as a method for predicting the properties of a composed software system. The premise behind compositional reasoning is that if we know the properties of the components our systems is composed of, we can define a function which tells us the properties of the system. Even though it might not be possible to define a rigorous and formal function, it should be possible to define the function in a less rigorous manner.

Stafford and Wallnau suggest that there are three questions that we should ask ourselves:

1. What are the quality attributes that we are interested in predicting?
2. What methods are there for reasoning about these quality attributes, and what properties do we need to know about the components?
3. How can we specify, measure and certify the properties of the components?

In the above mentioned paper, Stafford and Wallnau suggest that the used component technology can be used to constrain the construction of the system to designs that are easier to analyze, thus facilitating construction of software systems with predictable properties. They make an analogy to structured programming, comparing the restrictions of the component model to the restrictions of structured programming languages.

In another paper (Stafford and Wallnau, 2001b), Stafford and Wallnau discuss the need for certification of component properties. They explain that in order to be able to trust the predictions made, the developer needs to be able to trust information about the component properties.

In the case where the developer acquires components from an external source, the needed information might not exist, or it might not be reliable.

One solution that they propose is having a trusted third-party that certifies the component properties. Another solution is for the component to be packaged with a testing framework which allows the properties to be measured.

4. METHODS OF PREDICTING THE PERFORMANCE ATTRIBUTE

4.1 Prediction Enabled Component Technology

The research done at the Software Engineering Institute (Wallnau, 2003) suggests that it might be possible to incorporate prediction abilities into the component technology itself. This concept is called prediction enabled component technology (PECT).

The component model defines how components are allowed to interact, thus imposing constructive constraints on the composed system. The constructive constraints can restrict the architectural structure of the system.

Many properties can be expressed as a function of the properties of the components and the architectural structure of the system. By further restricting the architectural styles of the system, it is possible to use certain qualitative and quantitative theories for analyzing the properties of the composed system.

These theories are referred to as property theories. The restrictions made to the architectural structure of the system (in order to allow the property theories to be used) are referred to as analytic constraints.

According to this theory, it is possible to restrict the architecture of the composed system according to the constructive and analytic constraints, and thus achieve predictable behavior by construction.

The property theories define the component properties that need to be known in order to be able to make predictions about the properties of the assembled system.

The PECT concept developed by the Software Engineering Institute is an abstract concept that can be used to develop many different component technologies. A PECT consists of a construction framework and one or more reasoning frameworks.

The reasoning frameworks are linked to the construction framework by means of something called an interpretation.
An interpretation is a way to map assembly specifications to analyzable models.

The construction framework provides support for the construction of component-based systems. It consists of an abstract component technology, and tools that can be used when constructing component-based systems. The abstract component technology is a generalization of one or more component technologies. It defines the constructive and analytic constraints needed by the reasoning frameworks to provide valid predictions.

The reasoning frameworks provide support for prediction of the quality attributes of the composed systems. The reasoning frameworks consist of a property theory, and an automatic reasoning procedure. The property theory defines how to predict a certain property of a system based on the properties of its components. The automatic reasoning procedure is used to evaluate whether a claim about a property of the system is valid or not.

A few property theories have been developed at the Software Engineering Institute. One such property theory is presented in a technical report (Hissam et al, 2002). The property theory was developed to predict the average latency in a system with blocking and asynchronous interaction. The property theory was based on rate monotonic analysis (RMA). The property theory was empirically validated.

Another property theory is described in another technical report from the Software Engineering Institute (Hissam et al, 2004). In fact this property theory is a refinement of the theory mentioned above, in that it also allows prediction of the latency of tasks that are initiated aperiodically.

4.2 Component-Based Software Performance Engineering

Another approach to the problem of predictable assembly has been proposed by Bertolino and Mirandola (Bertolino and Mirandola 2003). They call their approach Component-Based Software Performance Engineering (CB-SPE). Their approach is focused on predicting only the performance and not any other quality attributes. Their approach is based on the discipline of Software Performance Engineering (SPE), which they have adapted to fit component based software development.

The SPE approach makes a distinction between the Software Model and the Machinery Model.

The Software Model is used to model the behavior of the software. The software behavior is represented by an Execution Graph. In an Execution Graph, the nodes represent software workload components, and the edges represent transfers of control. The nodes are weighted using a vector representing the resource demands of the component.

The Machinery Model on the other hand models the hardware, and it is based on Queuing Network models. In order to construct the Queuing Network model, information is needed about the components, their connections (topology), and some other important aspects (job classes, job routing etc.). The components and their connections can be derived from the system specification, while the other parameters can be found from execution graphs or from knowledge of resource capabilities.

The reason for separating the Software Model from the Machinery Model is that this allows us to combine different Software Models and Machinery Models, and e.g. predict the performance of an application (Software Model) in several different environments (Machinery Models), or vice versa.

The CB-SPE approach uses an UML notation based on the UML Profile for Schedulability, Performance, and Time (RT-UML). Bertolino and Mirandola explain that the RT-UML is a set of domain profiles for UML. The idea behind the RT-UML is to annotate the UML models with information that can be used to analyze certain aspects of the model (performance, real-time, schedulability, and concurrency).

According to Bertolino and Mirandola, the RT-UML can be used with many different analysis methods. The RT-UML is divided into a number of sub-profiles, which are specialized for certain analysis techniques. The CB-SPE approach uses the performance analysis (PA) sub-profile.

The CB-SPE is performed on two levels, the component level, and the application level.

On the component level, the goal is to obtain information about the components' performance attributes that can later be used when performing the analysis at the application level. The properties of the components are expressed in a way that is independent of the platform that the components are deployed on. This is done by defining an abstract model of the environment, running tests on the components in a concrete instantiation of the environment model, and expressing the obtained results as functions of the parameters of the environment. The components are then annotated with the parametrized properties.

On the application level, the goal is to predict the performance of the application based on the information about the component properties. The first step is to define the usage profile of the application. The usage profile consists of use cases weighted by their frequencies of use. The usage profile can be modeled as an Activity Diagram, where the use cases are annotated with their usage frequencies.

The second step is to choose the components to analyze. The parameters used to parametrize the components' properties should be known for the environment in which the components will be deployed, so it should be possible to obtain the property values of the components.

In the third step, the application's workflow should be described as a set of Sequence Diagrams, where the components are represented by objects, and the service requests or data exchange is represented by messages. A Deployment Diagram should also be constructed modeling the available resources and their characteristics. These diagrams should then be annotated with the proper performance values and parameters.
In the fourth step, a best-case analysis is performed, where the application is assumed to be the only one running in the execution environment, so that it has full use of all the resources of the environment. The result of the best-case analysis is an optimum bound on the application's performance. By comparing different assemblies, it should be possible to find some promising assemblies that deserve analysis in a more realistic setting.

In the fifth step, a more realistic performance model is used for the analysis. In this step, the environment is modeled using Queuing Network models. The authors claim that the Queuing Network models can be automatically obtained from the Execution Graph and the Deployment Diagram.

In the sixth step, the Queuing Network model is solved to obtain a set of performance properties for the assembly.

In the seventh and last step, the performance properties obtained from the analysis are evaluated to see if they fulfill the requirements of the system.

In another paper by the authors (Bertolino and Mirandola, 2004), they claim that steps four, five, and six can be fully automated, and that there are suitable tools to provide support for the other steps. They present a set of tools that can be used to support the CB-SPE approach. They note that some parts of the tools are still undergoing development.

The papers about CB-SPE do not include any claims as to which performance attributes the method can predict. There is also no reference to any direct validation experiments that the authors have done.

4.3 Other methods

In a paper by Yacoub (Yacoub, 2002), a method for analyzing the performance of a component-based application is presented. This method does not qualify as prediction, as it deals with the analysis of an already assembled system. However, the approach can be used to find bottlenecks of an already built system, and suggest where improvements could be made. The approach is based on instrumenting the application with logging statements whenever the application uses the services of a component. This instrumentation is added to the "glue code", which is assumed to define the workflow of the components. The approach also requires measuring of the environment on which the application is run (e.g. CPU load, memory use, and bandwidth). The measurements are done while using the system according to a defined scenario. By analyzing the measurements, bottlenecks can be found. The paper claims that the method can be used to analyze the response times, execution times, and resource utilization of the application.

Another paper (Liu et al, 2005), presents a way to predict the performance of a component-based application. The paper studies the performance in an EJB environment. The prediction model is based on queuing theory. The methodology consists of five phases.

In the first phase, a Queuing Network model is created for the component container. To do this, the main components of the container need to be identified, as well as the places where queuing delays occur. The details of the container components and their communication are abstracted, so that the model structure is simple. The model can be analyzed using standard queuing theory techniques.

In the second phase, the architecture pattern of the application is considered. The architecture consists of a set of components, their container attribute settings, and their communication pattern. The architectural pattern can be described as an activity diagram showing the steps performed to handle a request.

In the third phase, the architecture is analyzed to obtain a formula describing the service demand that the components places on the queues. This formula is parametrized with certain parameters representing the characteristics of the container.

In the fourth phase, a simple benchmark application is used to obtain the values of the parameters related to the container's characteristics. This results in a performance profile.

In the fifth step, the parameters of the performance model can be instantiated with the measured parameters from the performance profile, and the resulting queuing network model can then be solved using standard queuing theory.

The paper also shows an example of an EJB application which is analyzed using the methodology. The example shows how the different phases are performed to obtain a performance model for an EJB application (using a few different architecture variations).

The example also explains how a benchmark application is run on two different infrastructures (BEA WebLogic and Borland Enterprise Server) to obtain two different performance profiles, and how the prediction is then made. In the example, the performance attribute of interest is the average response time.

The performance was then measured from the application, and the measured performance was compared to the predicted performance. The error of prediction was within 11 percent.

5. DISCUSSION

As this paper shows, there are a few methodologies for predicting the performance attribute of component-based systems. This paper has mentioned the PECT approach from the Software Engineering Institute, the CB-SPE approach developed by Bertolino and Mirandola, as well as a queuing network based methodology developed by Liu et al.

The PECT approach is a general approach to predicting the quality attributes of a component-based system. It is not focused on any one particular quality attribute, but is rather a framework that can be plugged with different reasoning frameworks for different quality attributes.

The CB-SPE approach developed by Bertolino and Mirandola is focused on the performance attribute. It is based
on the discipline of Software Performance Engineering, adapted for component-based systems. The approach uses queuing theory to make its predictions.

The methodology of Liu et al is quite similar to the CB-SPE approach, as it is also built on queuing theory. In fact, the phases of this approach are in many ways the same as in the CB-SPE approach.

The PECT approach does not specify which attributes it can be used to predict. The possibility to predict a certain attribute depends on there being a reasoning framework for it. A few reasoning frameworks have been developed to predict the latency of a few different types of component-based applications.

The CB-SPE approach also does not specify which performance attributes it can be used to predict. It only states that it can be used to predict performance.

The methodology of Liu et al is used to predict the average response time of an EJB application. The authors don’t state whether the methodology could be used to predict any other attributes.

Since the methodology of Liu et al is quite similar to the CB-SPE approach, it is likely that they can be used to predict the same performance attributes.

The PECT approach has been validated at least for a few different performance attributes.

There is no indication whether the CB-SPE approach has been validated. Some tools have been developed to support the approach, so it is possible that some validation has been done as part of the development.

The methodology of Liu et al has been validated at least in one example case. The method was used to predict the average response time of an EJB application, and then the actual performance was measured and compared to the predicted performance.

It seems clear from the results obtained in this study that at least some performance attributes can be predicted for certain component-based systems.

In order for the prediction methods to work, the performance attributes of the components need to be known. It is, of course, possible to measure the component attributes, but that cannot be done before the components have been acquired. Therefore it is important that the component developers include this information in the component. Certification of the components might also be necessary, as the component developers might otherwise misrepresent the quality attributes of the component (either because of optimistic guessing, or intentionally to deceive).

Currently there is no standard for how to represent the components quality attributes. It seems likely that the different approaches to performance prediction need to get the component attributes in different forms (e.g. CB-SPE needs the performance attributes parametrized with the environment parameters). If there is no standard, component developers might not export the component attributes in the needed format. If the attributes are not exported in the needed format, prediction might become impossible.

The PECT idea of integrating the prediction technology into the component technology seems like the best way to get prediction technology used. The more automated the prediction can be, the more likely it is going to be used. If the prediction approach requires a lot of manual steps, and complicated model building, it will not be used as much. The CB-SPE approach of offering automation for some steps of the method, and tool support for the rest, also will likely lead to the method being used more.

The number of different performance attributes and component-based assemblies that can be predicted using the methods might still be too small for the methods to be generally useful. As more and more property theories are developed, the number of systems that can be predicted will increase, thus increasing the usefulness of the prediction methods.

6. SUMMARY AND CONCLUSION

The study shows that there exist some methods for predicting the performance of a component-based application. The methods can predict certain performance attributes for certain kinds of component-based applications. At least some of the prediction methods have been validated by experiment, and the results have been found to be reliable.

The methods might provide some benefit in certain contexts, but are probably not generally useful in an industrial setting.

As the methods are improved, and can be used to predict the performance of more and more systems, the methods will become more generally useful.

The integration of the prediction methods in the software development process, either through integrating it into the component technology or into the development environment in the form of tool support, will lower the threshold for taking the methods into use.

REFERENCES


