A QoS Driven Development Process Model for Component-Based Software Systems

Heiko Koziolek and Jens Happe

Graduate School Trustsoft *, University of Oldenburg, Germany
{heiko.koziolek, jens.happe}@informatik.uni-oldenburg.de

Abstract. Non-functional specifications of software components are considered an important asset in constructing dependable systems, since they enable early Quality of Service (QoS) evaluations. Several approaches for the QoS analysis of component-based software architectures have been introduced. However, most of these approaches do not consider the integration into the development process sufficiently. For example, they envision a pure bottom-up development or neglect that system architects do not have complete information for QoS analyses at their disposal. We extent an existing component-based development process model by Cheesman and Daniels to explicitly include early, model-based QoS analyses. Besides the system architect, we describe further involved roles. Exemplary for the performance domain, we analyse what information these roles can provide to construct a performance model of a software architecture.

1 Introduction

Quality of Service (QoS) analysis and prediction during early development stages of a software system is widely considered as an important factor for the construction of dependable and trustworthy systems. For component-based systems [10], the overall aim is to analyse QoS properties such as performance, reliability, availability, and safety based on specification documents of components and the architecture. For this purpose, compositional, analytical models can be constructed to allow predictions, even if the system only exists on paper. Using specifications of existing components, QoS predictions may be more precise than predictions for systems built from scratch.

To make early QoS analyses feasible in the IT industry, they have to become an integral part of the component-based development process. Cheesman and Daniels [2] describe a component-based development process model based on the Rational Unified Process (RUP). However, this approach does not contain any hint on how to include QoS analyses into the process.

On the other hand, several component-based QoS predictions approaches have been proposed. Most of these approaches focus on the analysis part and only contain very brief descriptions on how they are going to be integrated into the development process. For example, the component-based reliability [8, 9], performance [5, 1, 4], and safety [3] prediction approaches consider a pure bottom-up development where already existing components are assembled. This is a strong restriction, since combined top-down (starting from requirements) and bottom-up (starting from existing components) approaches as described in the following are more realistic.

* This work is supported by the German Research Foundation (DFG), grant GRK 1076/1.

I. Gorton et al. (Eds.): CBSE 2006, LNCS 4063, pp. 336–343, 2006.

All these approaches require a lot of additional information. QoS attributes of a component are not only determined by the component itself, but are influenced by the usage model, the deployment environment, its internal structure, and the services used by the component. In most cases, it is unclear how the information about all these factors is obtained and integrated. This is due to a lack of distinction concerning the roles and responsibilities during the development process.

The contribution of this position statement is an extension to the component-based development approach described by Cheesman and Daniels [2] to include QoS analyses. Augmenting the process model of the CB-SPE approach [1], we describe the responsibilities of the roles of component developer, system architect, system deployer, and domain experts. We discuss which information has to be provided by each role to construct a QoS prediction model. Our approach is not limited to performance analyses, but applicable to any QoS property.

This position statement is organised as follows. Section 2 introduces the roles in component-based development and discusses their responsibilities. Section 3 describes our QoS driven, component-based development process model and details on the specification and QoS analysis phases. In Section 4, we exemplary describe for a QoS property, which input values are needed for the construction of a performance model and associate these values with the roles discussed in Section 2. Conclusions follow in Section 5.

2 Roles in Component-Based Development

Since we want to evaluate QoS attributes at an early development stage, we need additional information about the internal component structure, the usage model, and the deployment environment. Not all of this information can be given by system architects themselves. Therefore, support of domain experts, component developers, and system deployers is required.

In our model, the *system architects* drive the development process. They design the software architecture and delegate work to other involved parties. Furthermore, they collect and integrate all information to perform QoS analyses and assemble the complete system from its parts. One of their information sources are the *domain experts*, who are involved in the requirements analysis, since they have special knowledge of the business domain. They are also familiar with the users' work habits and, thus, are responsible for analysing and describing the user behaviour.

On a more technical side, the *component developers* are responsible for the specification and implementation of components. They develop components for a market as well as on request. System architects may design architectures that are reusable in different deployment contexts. Sometimes, the actual deployment context is determined not until late development stages, especially if the software is developed for general markets. *System deployers* are responsible for specifying concrete execution environments with resources and connections. They also allocate components to resources. During the deployment stage of the development process, they are responsible for the installation, configuration, and start up of the application.

3 Integrating QoS Prediction into the Component-Based Development Process

In the following, the roles described in the former section are integrated into the component-based development process model featuring QoS analysis. We focus on the *development process* that is concerned with creating a working system from requirements and neglect the concurrent *management process* that is concerned with scheduling human resources and defining milestones. We base our model on the UML-centric development process model described by [2], which is itself based on the Rational Unified Process (RUP).

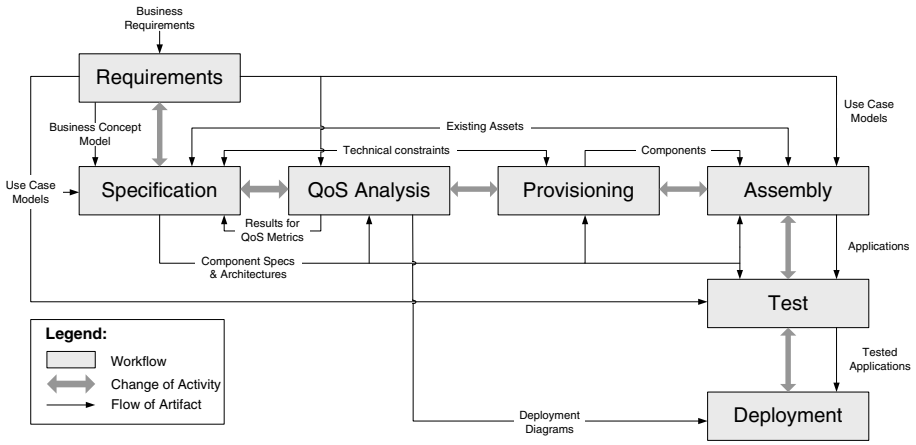

Fig. 1. Component-based Development Process Model with QoS Analysis

The main process is illustrated in Figure 1. Each box represents a workflow. The thick arrows between boxes represent a change of activity, while the thin arrows characterise the flow of artifacts between the workflows. The workflows do not have to be traversed linearly (i.e., no waterfall model). Backward steps into former workflows are allowed. The model also allows an incremental or iterative development based on prototypes. We have inherited the requirements, specification, provisioning, assembly, test, and deployment workflows from the original model and added the QoS analysis workflow. Component specifications, the architecture, and use case models are input to the QoS analysis workflow. Outputs of the QoS analysis are results for QoS metrics, which can be used during specification to adjust the architecture, and deployment diagrams that can be used during deployment.

In the following, we will only describe our extensions to the specification workflow and the new QoS analysis workflow. However, most of the other workflows are also influenced by QoS driven development. For example, a detailed description of the QoS requirements has to be compiled within requirements workflow. Furthermore, testing has not only to check functional properties, but also QoS attributes. For the other workflows and artifacts exchanged among them, we refer the interested reader to [2].

3.1 Specification Workflow

The specification workflow (see Figure 2, right column) is carried out by the system architect. The workflows of the system architect and the component developers influence each other. Existing components (e.g., from a repository) may have an impact on the component identification and specification workflow, as the system architect can reuse existing interfaces and specifications. Vice versa, newly specified components by the system architect can be input for the component requirements analysis of component developers, who design and implement new components.

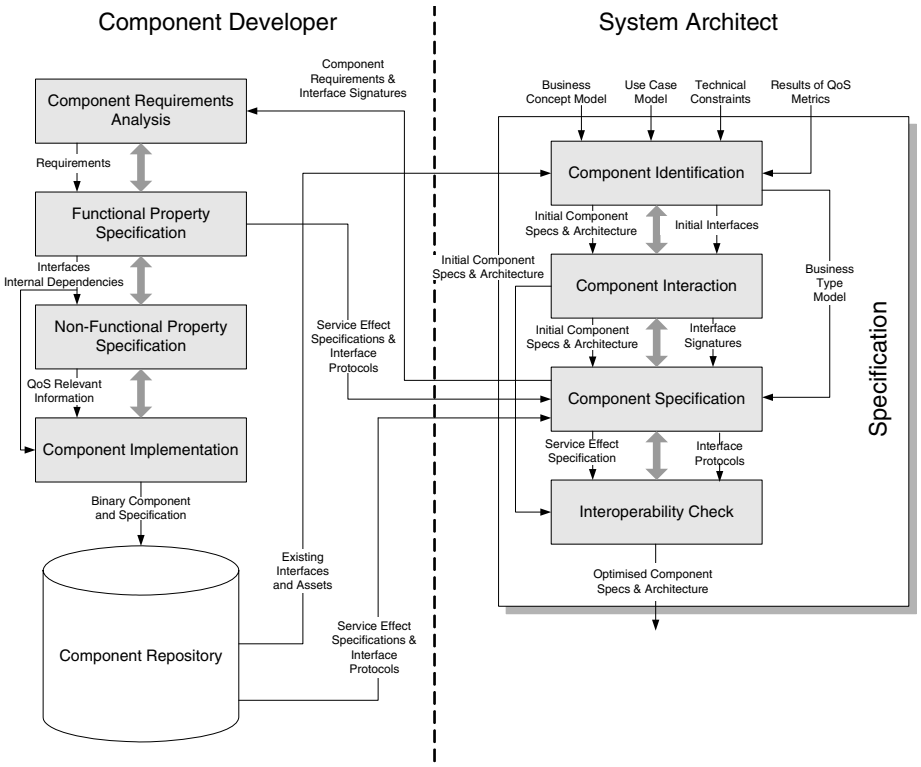

Fig. 2. Detailed View on the Specification Workflow

The workflows of the component developers are only sketched here, since they are performed separately from the system architect's workflows. If a new component needs to be developed, the workflow of the component developer (see Figure 2) can be assumed to be part of the provisioning workflow according to Cheesman and Daniels. Any development process model can be used to construct new components as long as functional and non-functional properties are specified properly. After the *component requirement analysis*, the *functional property specification* and then the *non-functional property specification* of the components follow. The functional properties consist of

interface specifications (i.e., signatures and protocols) and descriptions of internal dependencies between provided and required interfaces. We use service effect specifications from [7] to describe such dependencies. They model how a provided services calls its required services and can be specified by state machines. Non-functional, QoS relevant information includes resource demands, reliability values, data flow, and transition probabilities for service effect specifications. After *component implementation* according to the specifications, component developers may put the binary implementations and the specifications into repositories, where they can be retrieved and assessed by third party system architects.

The specification workflow of the system architect consists of four inner workflows. The first two workflows (*component identification* and *component interaction*) are adapted from [2] except that we explicitly model the influence on these workflows by existing components. During the *component specification*, the system architect additionally gets existing interface and service effect specifications as input. Both are transferred to the new workflow *interoperability check*. In this workflow, interoperability problems are solved and the architecture is optimised. For example, parametrised contracts, which are modelled as service effect specifications, can be computed [7]. The outputs of the specification workflow are an optimised architecture and component specifications with refined interfaces.

3.2 QoS Analysis Workflow

During QoS analysis, the software architecture is refined with information on the deployment context, the usage model, and the internal structure of components. Figure 3 shows the process in detail.

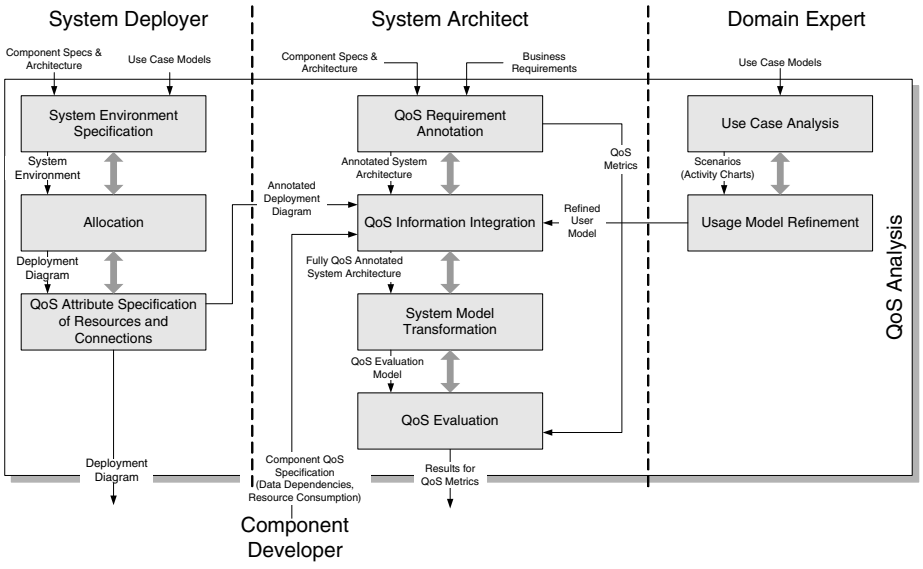

Fig. 3. Detailed View of the QoS Analysis Workflow

The system deployer starts with the *system environment specification* based on the software architecture and use case models. Given this information, the required hardware and software resources and their interconnections are derived. As a result, this workflow yields a deployment diagram that describes only the system environment without allocated components. The system deployer can also create a description of existing hardware and software resources. Moreover, a set of representative system environments can be designed if the deployment context is still unknown. During the *allocation*, the system deployer specifies the mapping of components to resources. The resulting deployment diagram is annotated with a detailed *QoS attribute specification* of the deployment environment. These specifications provide input parameters for the QoS analysis models used later. The resulting fully annotated deployment diagram is passed to the system architect.

The domain expert refines the use case models from the requirements during the *use case analysis*. A description of the scenarios for the users is created based on an external view of the current software architecture. The scenarios describe how users interact with the system and what dependencies exists in the process. For example, activity charts can be used to describe such scenarios. The scenario descriptions are input to the *usage model refinement*. The domain expert annotates the descriptions with, for example, branching probabilities, expected size of different user groups, expected workload, and user think times.

As the central role in QoS analysis, the system architect integrates the QoS relevant information, performs the evaluation, and delivers the feedback to all involved parties. In the *QoS requirement annotation* workflow, the system architect maps QoS requirements to the software architecture. For example, the maximum waiting time of a user becomes the upper limit of the response time of a component service. While doing so, the system architect specifies QoS metrics, like response time or probability of failure on demand, that are evaluated during later workflows.

During *QoS information integration*, the system architect collects the QoS specifications provided by the component developers, system deployers, and domain experts and integrates them into an overall QoS model of the system. This information is sufficient to transform the system and its behaviour into a stochastic process or simulation model as done in the *system model transformation*.

The *QoS evaluation* workflow either yields an analytical solution or the results of a simulation. QoS evaluation aims, for example, at testing the scalability of the architecture and at identifying bottlenecks. If the results show that the QoS requirements cannot be fulfilled with the current architecture, the system architect has to modify the specifications or renegotiate the requirements.

4 Information Required and Mapping to Roles

To construct a QoS prediction model, additional, extra-functional information besides the pure functional UML model is required. We will focus on information required for performance modelling as an example. The additional information needed to construct a performance model (e.g., a queueing network or stochastic Petri net) can be specified directly in UML with the SPT profile [6]. In this extension to UML, the performance analysis domain model describes the information needed to create a

performance model. It allows the inclusion of workload, component-behaviour, and resources into UML expressed as stereotypes and tagged values.

Domain experts are responsible for specifying all information closely related to the users of the system. This includes specifying workloads with user arrival rates or user populations and think times. In some cases, these values are already part of the requirement documents. If method parameter values have an influence on the QoS of the system, the domain experts may assist the system architect in characterising these values.

The *system deployer* provides information about the resources of the system (e.g., hardware-related like processing devices or software-related like thread pools). In the UML SPT profile, resources in deployment diagrams can be characterised as active or passive. Further attributes are scheduling policies, processing rates, or context switch times and must be specified by the system deployer. The system deployer is also responsible for adapting the platform independent resource demand specifications of the component developer to the properties of the system under analysis.

The *system architect* is responsible for extracting information from the requirements (e.g., maximal response times for use cases) and including them into the model. All information provided by the other roles are integrated by the system architect, who also has to estimate missing values. For example, the system architect might have to specify a parameter distribution for certain services if it influences the performance or he has to estimate the resource demand of components that have been provided without extra-functional specifications.

Component developers specify the performance of their components without knowledge where the components will be deployed, thus enabling independent third party performance analysis. First, they need to characterise the execution demands on the resources of the system in a platform independent way, for example, by specifying the number of processor or byte code instructions their services execute. The system deployer will use these values and parametrise them for the environment under analysis. Second, component developers have to specify how provided services call required services. This is necessary, so that the system architect can describe the control flow through the architecture. External calls to required services will be mapped to performance model steps in the UML SPT profile. The component developer can obtain these by code analysis or by evaluating design documents of the components.

For a performance analysis, transition probabilities and the number of loop iterations are required for calls from provided to required services. These values cannot be fully determined by the component developer, in case they are not fixed in the source code. Influences on these values may come from external sources, for example from the parameter values the service is called with or the results of external services. If such an influence exists, the component developer has to state this dependency in the component specification explicitly, so that the system architect can specify probability distributions for parameter values or exploit the postconditions of required services for this service.

5 Conclusions and Future Work

In this position statement, we have described how QoS analyses can be integrated into the early stages of a component-based development process. Several developer roles

participate in QoS analyses, as the system architects do not have all necessary information by themselves. We have demonstrated how component developers, system deployers, domain experts, and system architects interact during early QoS analyses. Additionally, we have described in detail, which information is necessary to conduct a performance analysis (exemplified by the UML SPT profile) and which of the roles can provide it.

A component-based development process model integrating QoS analysis is relevant for practitioners and researchers. Practitioners receive a recipe on how to tackle QoS problems during early development stages. Researchers are supported by showing a method on how to integrate their QoS analysis models into a practical development process.

Stating the specification responsibilities for the different values as in Section 4 is just a first step to an engineering approach to component-based performance prediction. Currently, we are looking for possibilities to retrieve some of the values from existing code (semi-) automatically. In this position statement, we have omitted an experimental evaluation of our development process model, which is planned for the future.

References

1. A. Bertolino and R. Mirandola. CB-SPE Tool: Putting Component-Based Performance Engineering into Practice. In *Component-Based Software Engineering*, volume 3054 of *Lecture Notes in Computer Science*, pages 233–248. Springer, 2004.
2. J. Cheesman and J. Daniels. *UML Components: A Simple Process for Specifying Component-based Software Systems*. Addison-Wesley, 2001.
3. L. Grunske, B. Kaiser, and Y. Papadopoulos. Model-Driven Safety Evaluation with State-Event-Based Component Failure annotations. In *Component-Based Software Engineering, 8th International Symposium, CBSE 2005, Proceedings*, volume 3489 of *Lecture Notes in Computer Science*, pages 33–48. Springer Verlag, 2005.
4. D. Hamlet, D. Mason, and D. Woit. *Properties of Software Systems Synthesized from Components*, volume 1, chapter Case Studies, pages 129–159. World Scientific Publishing Company, 2004.
5. S. A. Hissam, G. A. Moreno, J. A. Stafford, and K. C. Wallnau. Packaging Predictable Assembly. In *Proceedings of the IFIP/ACM Working Conference on Component Deployment (CD2002)*, pages 108–124, London, UK, 2002. Springer-Verlag.
6. Object Management Group OMG. UML Profile for Schedulability, Performance and Time. `http://www.omg.org/cgi-bin/doc?formal/2005-01-02`, 2005.
7. R. H. Reussner, I. H. Poernomo, and H. W. Schmidt. Reasoning on Software Architectures with Contractually Specified Components. In A. Cechich, M. Piattini, and A. Vallecillo, editors, *Component-Based Software Quality: Methods and Techniques*, number 2693 in Lecture Notes in Computer Science, pages 287–325. 2003.
8. R. H. Reussner, H. W. Schmidt, and I. H. Poernomo. Reliability Prediction for Component-Based Software Architectures. *Journal of Systems and Software*, 66(3):241–252, 2003.
9. R. Y. Shukla, P.A. Strooper, and D.A. Carrington. A Framework for Reliability Assessment of Software Components. In *Proceedings of the 7th International Symposium on Component-based Software Engineering (CBSE7), Edinburgh, UK*, volume 3054 of *Lecture Notes in Computer Science*, pages 272–279, 2004.
10. C. Szyperski, D. Gruntz, and S. Murer. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley, 2002.