A Software Development Process Supporting Non-functional Properties

Simone Röttger and Steffen Zschaler
Institute for Software and Multimedia Technology
Department of Computer Science
Dresden University of Technology
Dresden, Germany
email{Steffen.Zschaler, Simone.Roettger}@inf.tu-dresden.de

ABSTRACT
This paper presents a development process with particular focus on non-functional properties. The process is built around the concept of measurements at different levels of abstraction. It distinguishes the roles of measurement designer and application designer.

We propose the notion of multiple context models of measurements corresponding to different levels of abstraction. This enables us to provide CASE tool support for the refinement of non-functional specifications by requiring transformations between context models and measurement definitions to be specified by the measurement designer and using them to refine non-functional specifications.

KEY WORDS
Software Design and Development, Software Methodologies, Quality of Service, Component-Based Software

1 Introduction

Non-functional aspects of software systems – for example, Quality of Service (QoS) or security aspects – must be considered early in the system development to analyse or estimate the non-functional system behaviour. In the context of the COMQUAD[1] project we develop a methodology supporting the modelling of component-based systems with non-functional aspects.

The basic idea of existing development processes – especially MDA-based [1] approaches – is the refinement of system models from an abstract view of the system to a model close to the real implementation. This approach works in the context of non-functional aspects also. Our proposed process supports refinement steps for non-functional models of a system.

This paper gives an overview of a software development process which focuses on non-functional properties. In Section 2 we present the overall process, before Sections 3 and 4 take a closer look at the two main parts of the process: definition and usage of measurements, respectively. Finally, Section 5 summarises the key features and outlines future work.

[1]COMponents with QUantitative properties and ADaptivity at Technische Universität Dresden and Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany; supported by German Research Council; see also www.comquad.org

2 Overview of the Software Development Process

Figure 1 gives a graphical overview of our overall software development process for non-functional properties. A key feature of our approach is the separation of measurement definition from measurement usage, i.e., specification of non-functional properties of applications using those measurements. A *measurement* is something that can be applied to a system and yields a quality value for the system being measured. Examples of measurements are response time, delay, etc. Measurements are always defined relative to a *context model*, which describes the features of an application system that must be known in order to establish the value of a measurement. Thus, measurements can be defined independently of a concrete application system. To apply measurements to an application system, a mapping between the *component model* of the application system and the context model of the measurement must be defined. Measurement definitions can be very complex, but on the other hand will be developed only once. Therefore, we separate the roles of *measurement designer* and *application designer* in our process. Their combined efforts lead to a specification of the system including its non-functional properties.

The measurement designer uses a graphical notation based on the specification language CQML$^+$ [2] and ideas from [3] to specify measurements and their context models. He can define different context models representing different levels of abstractions. Such context models are connected via *transformation specifications*.

The application designer is concerned with refining a system model of the application under development. He uses the measurements and transformations defined by the measurement designer. We have defined a graphical notation for QoS constraints which allows us to represent non-functional specifications in UML models. In that notation, any element of the UML model can have attached a con-

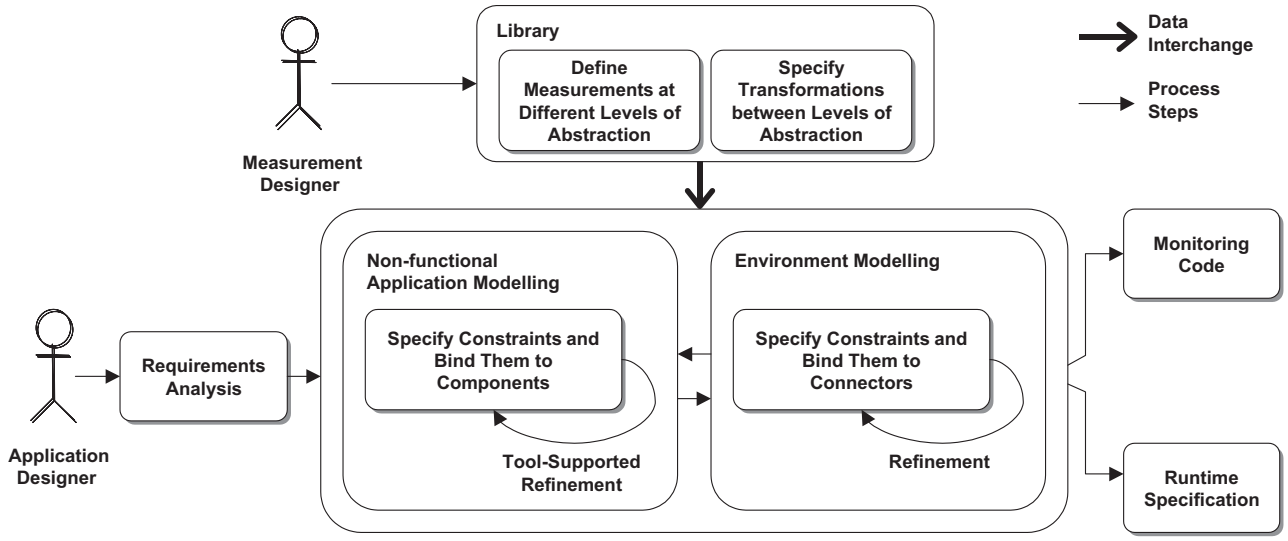

Figure 1. Overview of the development process for non-functional properties.

straint over predefined measurements.

For functional modelling we primarily use the component model element of UML 2.0 [4] extended with a stereotype for interfaces to be able to distinguish between operational and streaming interfaces.

The resulting non-functional specification can be used for a variety of purposes. Besides generating code for runtime monitoring of QoS parameters, its main use lies in providing a base for resource reservation and QoS management in the running system. Therefore, the specification is first transformed from the human understandable form of CQML$^+$ to a more compact, XML-based form that is optimized for machine usage. The runtime environment – the component container – then uses this specification to reserve resources for each component.

The following two sections take a closer look at the two main parts of the process: definition and usage of measurements. Figure 2 shows a close-up of these steps.

3 Definition of Measurements

The measurement designer uses a CQML$^+$ based graphical notation to write measurement definitions. Each measurement definition is written relative to what Aagedal calls a computational model [5]. We prefer the term *context model*, as it is really a model of the context of a measurement definition – that is, it comprises the elements necessary for specifying the semantics of the measurement. For each context model and each component model to be used there must exist a mapping from the concepts of the component model to concepts of the context model. These mappings are application independent, but they depend on the component model. The EJB component [6] model is an example of what we mean by the term *component model*.

It is helpful for the application designer to use measurement definitions at a level of abstraction that is appropriate for the current state of development. To enable this, the measurement designer needs to explicitly define context models at those levels of abstraction. In effect, each context model represents the specification of a specific level of abstraction. This is different from the proposal in [5], where a single computational model is used independent of level of abstraction.

Additionally, the measurement designer defines the relationships between context models representing different levels of abstraction by writing transformation specifications. A tool can then use these specifications to automatically transform measurement definitions.

The measurement definitions, context models and transformation specifications are later used by the application designer to create and refine non-functional models of an application. This part of the process will be described in the next section.

4 Usage of Measurements

For application development, we propose a process based on the measurement definitions at different levels of abstraction that were previously created by the measurement designer. This approach supports a notion of refinement of non-functional constraints, thereby allowing application designers to specify abstract non-functional properties very early in the development cycle and refining them as the functional model becomes more evolved. The process is divided into the following steps:

1. *Modelling non-functional properties of the application:* The application designer constrains measure-

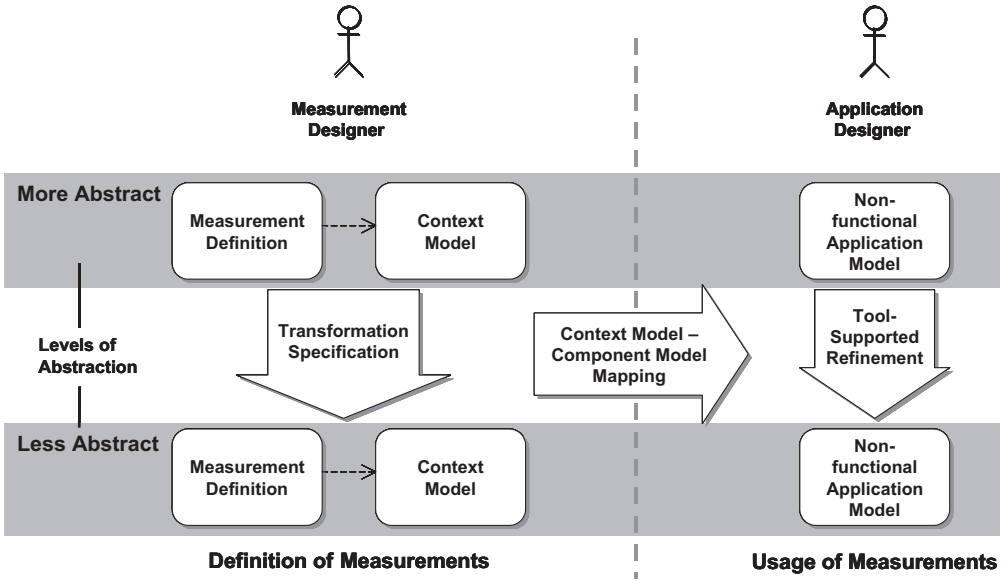

Figure 2. Close-up on definition and usage of measurements.

ments and binds these constraints to elements of the functional application model.

2. *Tool-supported refinement of the non-functional constraints:* The application designer chooses one out of different kinds of provided refined measurements. These have been previously provided by the measurement designer together with a textual description of each measurement.

3. *Modelling non-functional requirements on the runtime environment:* The application designer uses connectors to model the influence of the container on non-functional properties of the application.

We will explain these steps in some more detail in the following subsections:

4.1 Modelling Non-functional Properties of the Application

The application designer uses a graphical notation to attach constraints over measurements provided by the measurement designer to elements of the application model. The CASE tool can check whether a measurement is applicable to a model element by using the mapping between the measurement's context model and the application's component model. The measurement designer has previously provided such mappings for every context model (see Section 3).

The CASE tool may further perform analysis, such as checking whether a non-functional property provided by a server component satisfices the requirements of a client component.

4.2 Tool-Supported Refinement of Non-functional Constraints

As application design progresses it may become necessary to refine the non-functional specification. The application designer selects a measurement constraint to refine and the CASE tool provides a list of target context models that are more detailed than the context model currently associated with the measurement under refinement. After the application designer selects the appropriate refining context model, the tool prompts him with the possible choices for refinement of the non-functional constraint.

4.3 Modelling Non-functional Requirements on the Runtime Environment

The application designer uses connectors to model communication channels between components. The concept of connectors has originally been introduced in the context of Architecture Description Languages (ADLs) [7]. The concept can be extended to allow the application modeller to include QoS properties of the runtime environment into the application model. Thus, such properties can be taken into consideration in QoS analysis of the application model.

5 Conclusion

Non-functional properties must be considered throughout the development cycle of an application system and at different levels of abstraction. We have shown a software development process which supports different levels of ab-

straction not only for functional models, but also for non-functional specifications.

In the context of non-functional properties, the roles of measurement designer and application designer (that is "measurement user") can be distinguished. In this paper, we have proposed a software development process that uses this distinction together with explicitly defined context models of measurements to provide CASE support for refinement of non-functional specifications of systems.

It is important to point out, that the process is generic. For any one measurement there could be any number of context models and, correspondingly, any number of different levels of abstraction. It remains a question still to be answered how this large amount of models can be managed in a way that further reduces the complexity for the application designer and makes choosing the next model for refinement easy.

We are currently working on a tool prototype for our development process. This prototype is going to support the concepts presented in this paper. Before we can build the prototype, however, some more work needs to be done: We need to specify the precise format of context models as well as the language used to define transformations between context models.

References

[1] Anneke Kleppe, Jos Warmer, and Wim Bast. *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison Wesley Professional, April 2003.

[2] Simone Röttger and Steffen Zschaler. CQML$^+$: Enhancements to CQML. In *Proceedings of the 1st International Workshop on Quality of Service in Component-Based Software Engineering*, pages 43–56, Toulouse, France, June 2003. Cépaduès-Éditions.

[3] UML profile for modeling quality of service and fault tolerance characteristics and mechanisms: Revised submission, May 2003. Submitted by: I-Logix Inc., Open-IT, THALES; OMG document number realtime/03-05-02.

[4] Unified modeling language: Superstructure version 2.0, July 2003. OMG, document number ptc/03-07-06.

[5] Jan Øyvind Aagedal. *Quality of Service Support in Development of Distributed Systems*. PhD thesis, University of Oslo, 2001.

[6] Sun Microsystems. Enterprise JavaBeans Specification, version 2.0. Final Release, August 2001.

[7] Nenad Medvidovic and Richard N. Taylor. A framework for classifying and comparing architecture description languages. In *Proceedings of the Sixth European Software Engineering Conference together with the Fifth ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC-FSE97)*, pages 60–76, Zurich, Switzerland, September 1997.