Automating Systems QoS Design using Model Transformations

Amogh Kavimandan and Aniruddha Gokhale
Dept. of EECS, Vanderbilt University
Nashville, TN
{amoghk, gokhale}@dre.vanderbilt.edu

ABSTRACT

We describe Quality of service pICKER (QUICKER), a model-driven QoS mapping toolchain for supporting the design and evolution of application QoS. QUICKER automates the mapping of QoS requirements onto platform-specific QoS configuration options by (1) choosing appropriate subset of QoS options for given QoS policies and (2) assigning values to each of these selected QoS options. QUICKER also provides support for validating the generated QoS configurations and resolving any dependencies between them.

Categories and Subject Descriptors

D.1.2 [**Automatic Programming**]: Program Transformation

General Terms

Middleware Configuration Design

Keywords

Model-driven development, model tansformations, component middleware, web services, SOA, CCM

1. INTRODUCTION

Service Oriented Archictures (SOA) such as Web Services (WS) and Component middleware technologies such as Enterprise Java Beans (EJB) and CORBA Component Model (CCM) have raised the level of abstraction for the application developers by separating functional and non-functional aspects during application software development lifecycle. Such *systems software* technologies can be used by the application developers to (1) allocate CPU, network and Operating System (OS) resources *a priori*, for various system building blocks, (2) specify event registration, delivery, and filtering mechanisms for asynchronous communications between the application functional building blocks, (3) (re-)configure, and (re-)deploy target applications, and (4) marshal/demarshal communication requests, configure transactional and persistence properties of applications. In an effort to support a wider range of target application domains, these systems software technologies have evolved into a highly configurable and customizable systems software platforms that provide a number of configuration mechanisms to satisfy non-functional requirements of applications in each of these target domains. Owing to such a flexibility however, the size of the system software configuration space (*i.e.*, the configuration mechanisms suited for an application and their appropriate value set) becomes large. A comprehensive knowledge of various configuration options, their inter-dependencies, and how they impact application-level requirements is critical to correctly perform configuration of the systems platform. Failure to carefully map policies to low-level configuration options will lead to a sub-optimal system software configuration degrading the overall application performance, or worst run-time errors that are costly and difficult to debug.

This poses a significant challenge for application developers who are seldom experts at performing optimal configuration of the systems platform for their application.

Our Solution. We have developed QUality of Service pICKER (QUICKER), a model-driven engineering (MDE) QoS mapping toolchain to support QoS design of the implementation systems software(s). QUICKER provides a system composition language to enable application developers to annotate applications with QoS policies *i.e.*, QoS requirements of applications. The current implementation of QUICKER supports QoS configuration of applications implemented using CCM and WS systems software platforms. It also defines model transformations to automatically map these application QoS policies to a set of platform-specific QoS configuration options that are required to satisfy the specified QoS policies. Finally, QUICKER uses generative techniques to synthesize model-checking input of the application to verify various QoS options generated through model transformations.

2. MODEL-DRIVEN QoS MAPPING FOR SYSTEMS SOFTWARE PLATFORM

Figure 1 shows the model-driven QUICKER toolchain for supporting the design and evolution of application QoS. We discuss various capabilities of the toolchain in this section.

Challenge 1: Capturing application QoS policies. As already stated, application developers are domain experts with a thorough understanding of the application business logic but often lack the knowledge about QoS configuration

ASE '07 November 5-9, 2007, Atlanta, Georgia.

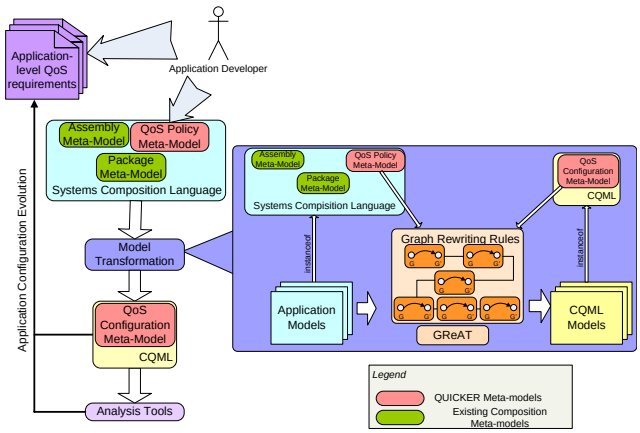

Figure 1: Model-driven QUality of service pICKER (QUICKER) Toolchain

space of the systems platform to optimally configure the systems platform for their respective applications.

Solution: Domain-specific QoS policy specification using QUICKER. We have developed a QoS Policy DSML that allows specification of desired application QoS policies. The QoS Policy meta-model shown in Figure 1 models the QoS Policy DSML, and has the semantics of the application QoS requirements. By focusing on *what* is expected from the application (*i.e.*, application QoS requirements) rather than *how* the application QoS may be achieved (*i.e.*, low-level platform-specific QoS options), QUICKER enables easier and intuitive application QoS specification.

Challenge 2: Identifying the set of platform-specific QoS options from application policies. Once application QoS policies are captured using the QoS Policy DSML, these policies still need to be mapped onto correct platform-specific QoS options. Current solutions to resolve this challenge are ad-hoc, *i.e.*, manually identifying the QoS options from the given application QoS policies. An application typically goes through several iterations during its software development cycle (and possibly, during its maintenance cycle, in order to incorporate new requirements). Without automated tool support, particularly for large-scale applications, it is time consuming, error prone and in some cases infeasible for application developers to correctly configure the systems software for a given application QoS policy set.

Solution: Automated QoS policy mapping through model transformations. We have defined transformation algorithms using GReAT [1] toolchain that translate the QoS policies into detailed, platform-specific QoS configuration options. QUICKER model transformations define rules that perform the following activities: (1) choosing an appropriate subset of QoS options that high-level application QoS policies map to, and (2) choosing valid values for each of these QoS options to perform *QoS configuration* of the systems platform. As shown in Figure 1, transformations are defined in terms of meta-models, and thus can be used repeatedly for any application models that conform to the QoS Policy DSML. The generated QoS options are themselves models to allow for further analysis/transformations.

Challenge 3: Validating platform-specific QoS options. QoS options for an application may be associated at various levels of granularity at the implementation systems platform. For example, RT-CCM configuration options have component-level associations, RT event channel service options have asynchronous connection-level associations, and WS Reliable Messaging options have port-level (*i.e.*, event source and/or event sink) associations. Depending on their associations, QoS options are often dependent on each other and hence a change in value of one QoS option may affect many other QoS options [2]. Thus, such dependencies must be resolved before the application can be prepared for deployment. Again, manual approaches to resolving such a dependency are limited in their applicability and do not scale as the size of the application increases.

Solution: Use model-checking to validate (generated) QoS options. QUICKER extends the Bogor Input Representation (BIR) [3] with new constructs that enable specification and model-checking of system properties more closely to the domain of implementation platform. Using these extensions, a systems platform-based application can be expressed in terms of BIR and the properties of the application(*i.e.*, QoS options) can be validated using the model-checking framework. QUICKER uses generative techniques on the models of QoS options in order to synthesize: (1) input to Bogor model-checking framework in order to model-check the QoS options, and (2) descriptors in middleware-specific format that are required to configure application QoS before deployment.

3. CONCLUDING REMARKS

Ad-hoc, manual approaches to mapping application QoS to valid platform-specific QoS options are tedious, error-prone and do not scale with the size of application. In this paper we discussed QUICKER, a model-to-model transformation toolchain that provides an automated, scalable, and reusable approach to the QoS mapping challenge. QUICKER provides intuitive modeling abstractions to facilitate application QoS policy specification and model transformation algorithms that map these QoS policies to the platform-specific QoS options that will ultimately achieve the desired application QoS. QUICKER is available as open-source from `www.dre.vanderbilt.edu/CoSMIC/`.

4. REFERENCES

[1] G. Karsai, A. Agrawal, F. Shi, and J. Sprinkle. On the Use of Graph Transformation in the Formal Specification of Model Interpreters. *Journal of Universal Computer Science*, 9(11):1296–1321, 2003.

[2] Amogh Kavimandan, Krishnakumar Balasubramanian, Nishanth Shankaran, Aniruddha Gokhale, and Douglas C. Schmidt. QUICKER: A Model-driven QoS Mapping Tool for QoS-enabled Component Middleware. In *Proceedings of the 10th IEEE International Symposium on Object/Component/Service-oriented Real-time Distributed Computing*, Santorini Island, Greece, May 2007.

[3] Robby, Matthew Dwyer, and John Hatcliff. Bogor: An Extensible and Highly-Modular Model Checking Framework. In *Proceedings of the* 4^{th} *Joint Meeting of the European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 2003)*, Helsinki, Finland, September 2003. ACM.

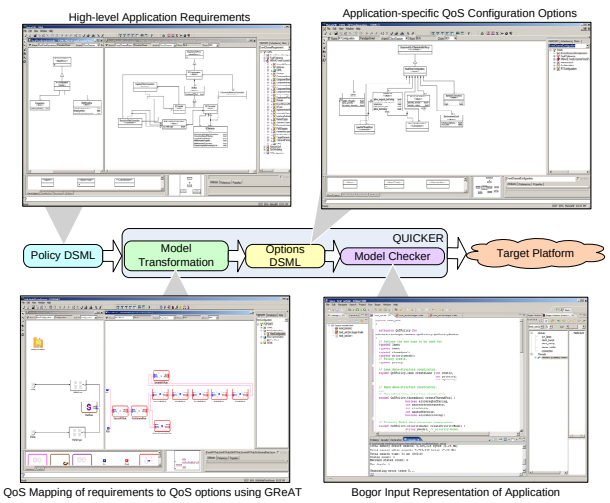

Figure 2: QUality of service pICKER (QUICKER)

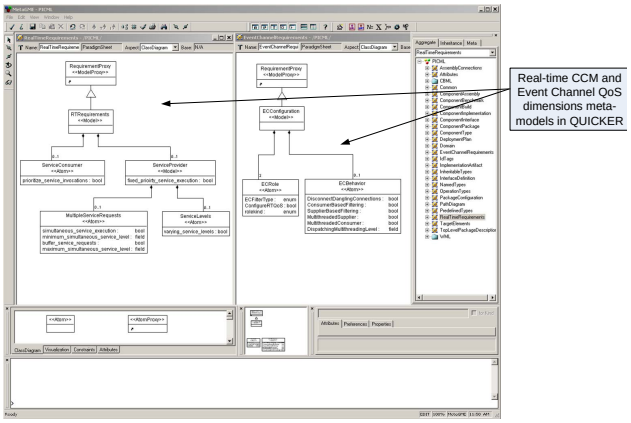

Figure 3: QoS Policy DSML

APPENDIX

A. DEMONSTRATION DETAILS

Our demonstration will use a representative application scenario to show the automation capabilities of QUICKER open-source model-driven toolchain. The QUICKER toolchain and the overall QoS mapping process is shown in Figure 2. In particular, our demonstration will illustrate the following three functionalities of the toolchain:

- QoS Policy domain specific modeling language (DSML) which is used by the application developer to describe the application QoS requirements.
- QoS Policy Mapping which translates the user-specified application QoS requirements into middleware-specific configuration options.
- Generative capabilities in QUICKER which generate model-checking input from the application model and deployment artifacts that are required to ultimately run the application using runtime infrastructure of middleware platform(s).

A.1 Modeling Capabilities in QUICKER

Firstly in this section of the demonstration, in order to motivate the need for a model-driven approach to QoS mapping, we will explain the application scenario used, and why automation of QoS design and evolution for such a representative scenario is critical. We will give a brief overview of Generic Modeling Environment (GME) that is used as the meta-programmable framework for development of our QoS Policy DSML in QUICKER.

Next, we will explain the DSML shown in Figure 3 and its support for modeling QoS requirements in various QoS dimensions (such as Real-time CCM, event channel service, WS-notification, WS-Reliable Messaging) and middleware platforms (such as Web Wervices and CCM). We will demonstrate these modeling capabilities by developing a QoS model of the application that captures its QoS requirements completely in one QoS dimension. For example, Figure 4 shows Real-time CCM requirements of an application modeled using QoS Policy DSML.

Finally, we will compare this model against QoS specification using low-level, platform-specific configuration options (either using a DSML, or textual declarative XML-like notation) to show that modeling application QoS using QUICKER is considerably simpler than existing approaches.

A.2 Automated Translation of QoS requirements to platform-specific QoS Options

In this section, We will give an overview of Graph Transformation and Rewriting (GReAT) model transformation toolchain used for automated mapping of QoS requirements. We will also explain the challenges involved in mapping application-level QoS requirements to middleware mechanisms, *i.e.*, middleware configuration options required to realize these requirements.

Next, we will show how QUICKER uses model transformations defined using GReAT that automatically translate application-level QoS policies into a subset of implementation platform-specific QoS options. This translation involves not only choosing the right set of QoS options from application requirements but also using appropriate values for these options such that the requirements can be satisfied. We explain a sample transformation rule as shown in Figure 5 in order to exemplify the semantic translation of a specific application QoS policy into a (set of) platform QoS option(s). QUICKER model transformations generate QoS options as models as shown in Figure 6, such that they can be used further by other analysis/transformation tools.

A.3 Generative Capabilities in QUICKER

Finally, the demonstration presents the generative capabilities of QUICKER. QUICKER Model interpreters developed for QoS Policy DSML are used in this step to synthesize:

- Input to Bogor model-checking framework. As part of the QUICKER toolchain, we have developed extensions to Bogor that allow application specification and model-check application properties more closely to implementation middleware, we have defined composite constructs in Bogor that represent application structure and properties (*i.e.*, QoS options) as though they were native BIR constructs.
- Deployment Descriptors in middleware-specific format. In preparation of application deployment, the runtime

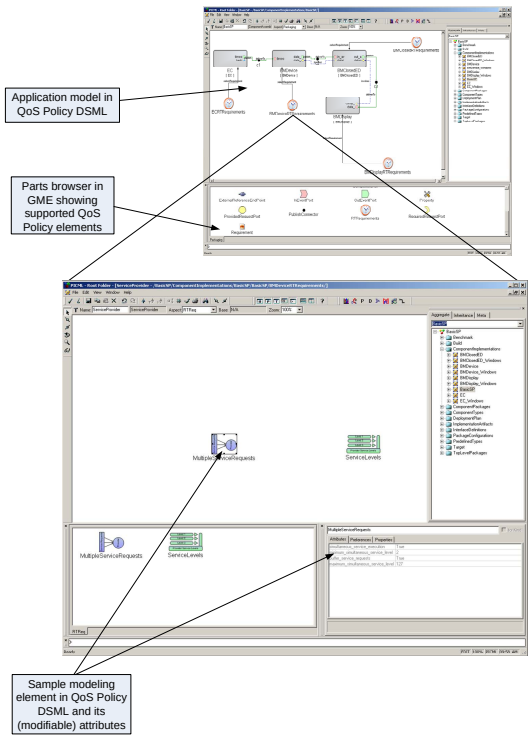

Figure 4: Application Model specifying Real-time CCM Requirements

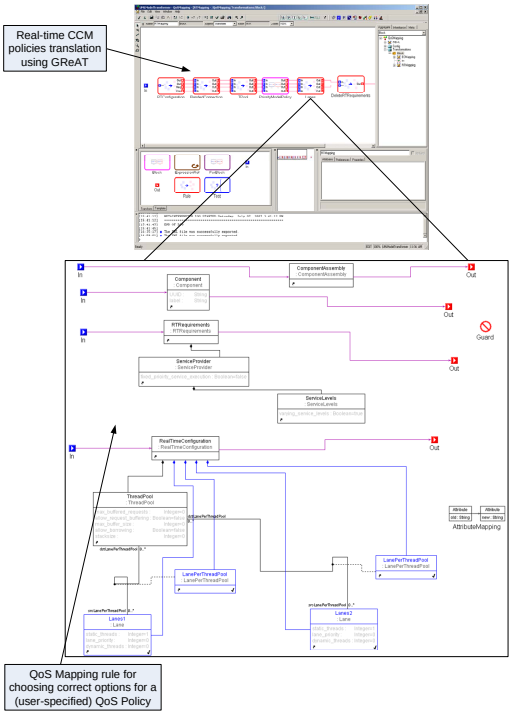

Figure 5: GReAT-based Model Transformations for QoS Policy Translation

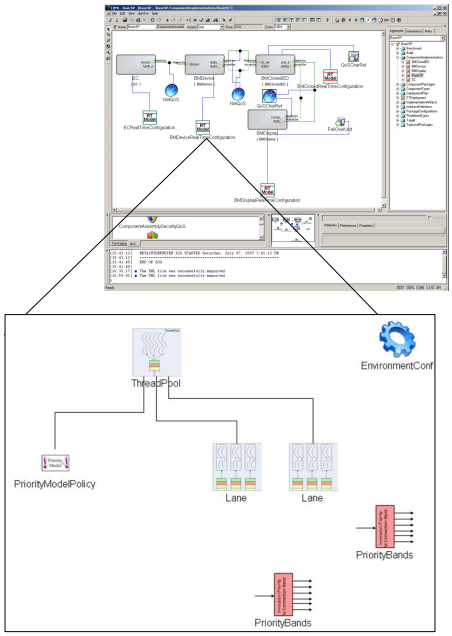

Figure 6: Transformation-generated Platform-specific QoS Options

infrastructure of the middleware would use these application descriptors such that application QoS can be configured.

The complete QUICKER QoS Policy DSMLs, GReAT-based model transformations, and model interpreters are part of the CoSMIC toolchain and are available as open-source from `http://www.dre.vanderbilt.edu/CoSMIC`.