Quality-of-Service Modeling and Analysis of Dependable Application Models

András Balogh and András Pataricza

Budapest University of Technology and Economics,
Department of Measurement and Information Systems
H-1117 Budapest, Magyar Tudósok krt. 2.
{abalogh,pataric}@mit.bme.hu

Abstract. We present an approach for Quality-of-Service (QoS) aware model-driven development of distributes applications. Our methodology includes platform- independent and platform-specific modeling of software components and the modeling of the hardware platform. We also introduce modeling and analysis of non-Functional component and system properties to improve the quality of software products and to increase the productivity of deployment by the early recognition of possible problems and bottlenecks in performance, dependability, and other non-functional aspects.

1 Introduction

Nowadays, a new trend can be observed in multiple software development domains. The model-driven development has become an industrial practise in both business system development and embedded system development, as on many other fields of computing.

With the announcement of the Model-Driven Architecture (MDA) paradigm [12] of the Object Management Group (OMG) a new approach and a conceptual framework have been established for the development using modeling and code generation. Based on the experiences it is now clear that the original MDA approach focusing on the functional properties of systems is not enough in itself. Many application domains need other perspectives and tools to solve development problems.

In the business application domain the dependability of business systems (E-Commerce, Customer Relationship Management) have become a critical issue in the past years. Similarly, in the embedded systems domain dependability of systems controlling vehicles or industrial processes is a key issue.

While QoS (quality of service) properties play a key role in these systems, they are usually neglected during the system development and they are also not involved in the traditional MDA workflow. Typically, the QoS assessment takes place only in the late integration testing phase. This means that the potential QoS problems are not handled in the early phases of system design, which can result in dramatic cost increase (in case of system redesign), or inadequate service quality of the system.

To avoid such risks, we suggest using formal analysis techniques during the early system design phase to assess the key quality aspects of the systems. This allows the early recognition of potential problems and the avoidance of them by modifying the system design before the implementation phase. This results in lower development costs and time, because the QoS problems can be eliminated in much earlier phases of development than in the current practise.

In this paper we propose a modeling notation for distributed systems that is based on UML 2.0 [13] and includes functional and non-functional properties of both the software and hardware components. In Sec. 2 we introduce our primary application domains, in Sec. 3 and Sec. 4 the analysis and implementation level modeling is discussed, and Sec. 5 demonstrates our model analysis approach.

2 Target Domains

In this section we introduce two different domains, those fundamental structure is very similar; therefore the methods introduced later can be used for the analysis in both application area.

2.1 Business Applications

Nowadays business systems are built according to the Service-Oriented Architecture (SOA). The applications are composed of stand-alone loosely coupled components (called services) that interact each other using message interchange. These services are deployed on heterogeneous hardware and software infrastructure that can be distributed around the world.

The stand-alone services can be composed either using peer-to-peer connections, or hierarchically, using an orchestration defined using a standard language, like Business Process Execution Language [6]. The later enables the usage of Commercially Off-The Shelf (COTS) service components even if the user only knows the interface specification of the particular service (specified by a standard description like WSDL [1]). This results in a black-box service composition between independent parties.

The QoS analysis of these composed systems is important to determine the overall QoS parameters of the system based on the component characteristics. The component properties (dependability, performance, cost) are usually specified by the service provider in a standard format like Web Service Level Agreements - WSLA [2]).

2.2 Distributed Embedded Systems

In the embedded systems domain, especially in the automotive and aerospace areas the current trend is the integration of independent applications on the same hardware nodes to decrease the number of Electrical Control Units (ECU), and the system cost. This trend results in a system that runs both safety critical

subsystems (like steer-by-wire, break-by-wire) and non-safety critical subsystems (window lifter, entertainment).

The various subsystems must be separated in the spatial and temporal domain to avoid interaction between them. This must be ensured by the execution platform [16]. However, the correct execution of subsystems is guaranteed only if the developer can configure the execution platform to get sufficient computing and communication resources to all of the jobs (stand-alone software units) that must be executed in the system. This requires the analysis of system models and the QoS-aware development of integrated applications.

Both application domains introduced in this section share several common properties. The systems are distributed, and they contain several communicating atomic components or jobs that can share the same hardware resources. The QoS analysis has to determine the system-level non-functional characteristics of the systems based on the models and the component-level QoS attributes.

3 Analysis Models

In our view, the analysis - or platform independent - model of the application contains the main logical components (or functional blocks) of the system under design, and also the key QoS requirements defined for the system. QoS requirements can be performance, schedulability, dependability, or security related depending on the target application domain.

3.1 Running Example

We will use an example application to illustrate the techniques introduced in this paper. This example is a simple embedded application that controls the window lifter in a car. The system contains three functional blocks, a keypad, a control logic, and the window lifter motor. The user can initiate window movements using the keypad, the control logic generates commands for the actuator and the actuator moves the window. There is also a feedback from the motor to the control logic to signal over-current conditions (possibly end state or jam).

3.2 Service Components

Designers usually work with the top-down approach, that starts from a high abstraction level and drills down to the details of the system. Using this technique, the first diagram we should create is the high level component diagram of the system (Fig. 1). The diagram contains the main functional blocks of the application and their interfaces for the communication.

In the embedded domain we also need a method for modeling the interaction of the system with the physical world. This is done via *sensor* and *actuator* artifacts. For instance the keypad sensor component uses a sensor called *KeypadSensorResource*. These elements are requirements for the target platform: the component needs a hardware element that can act as a keypad.

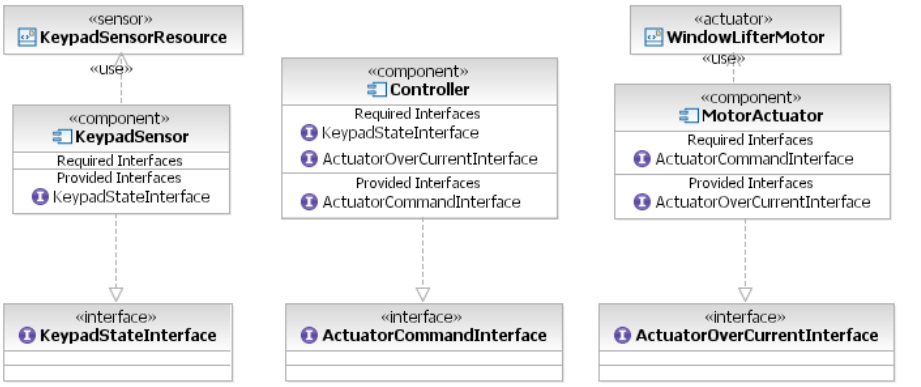

Fig. 1. The components of the example system

3.3 Detailed modeling

Fig. 2. Detailed model of a component

During the detailed modeling designers refine the initial component model and create a more detailed structural model for the components. In this phase not only the internal parts but also the communication interfaces can be designed in more detail (Fig 2).

The components are divided to *job*s, that are basic function blocks. A job can use resources and has *interface*s. Jobs are communicating with each other via *message*s. This design step helps the designer to clearly identify and separate sub-functionalities that can be implemented in a loosely-coupled way. Each basic function block can be treaded as the provider of a specific service. The application itself is then a composition of services. This idea is the same as of the Service-Oriented Architecture approach.

The messages - in the platform independent level - mark only the fact of communication between functional elements, and does not say anything about the

method of communication. The message specifies only the sender, the receiver(s), and the data that is transmitted.

For instance in Fig. 2 we can see the refined structural model of the KeypadSensor component. In our case the component consists of a single job that is using the sensor resource (as on component level) and provides a message (KeypadStateMessage) through its interface (KeypadStateInterface). The message contains an integer value containing the code of the button pressed on the keypad.

In case of complex models the usual UML diagrams can be used for the description of internal job behavior (statecharts, activity diagrams), or for the description of typical communication schemes (sequence diagrams). Our proposed notation has extension only for the structural modeling as this is the most relevant part of the model from the current analysis point of view.

3.4 QoS Attributes in The Analysis Model

Our modeling approach also incorporates the non-functional or Quality-of-Service (QoS) attributes even in the analysis models. These measures are usually platform independent, so they can be defined on this level. There exist standard UML profiles for modeling non-functional attributes of systems [14] and [15], and we suggest using them, but in a more compact, domain-specific form.

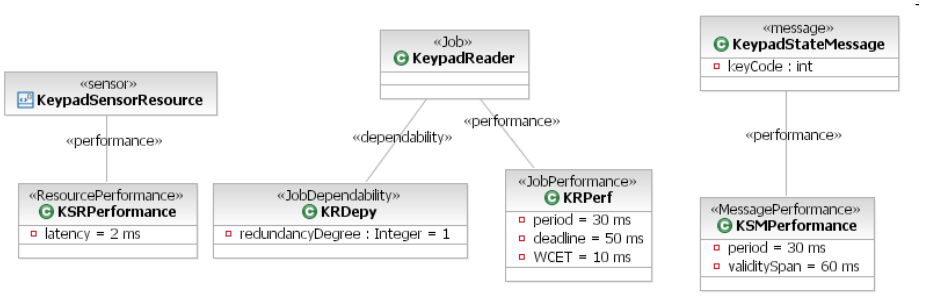

Fig. 3. QoS values in the model

Based on the specialities of the target domain and the aspects we wanted to analyse (performance, dependability) we aggregated the important measures to a single class per aspect. This technique allows the creation of custom QoS profiles for UML modeling that suit the needs of the current application domain and are more compact than the standard profiles.

The modeling notation is illustrated by Fig. 3 that shows the QoS values defined for the sensor component. It has performance-related attributes for the sensor hardware (maximum latency), for the job (period, deadline, worst-case execution time), and also for the message (period, validity span). These values

define requirements: the message has to be transmitted each 30 ms, its value is valid for 60 ms, and so on.

Besides these local requirements usually there are also more complex ones. For instance, in the example system the most important measure is the overall time from pressing a button to the start of the window movement (must be less than 150 ms in our example). These complex measures can also defined in the model using expressions on the model elements. For instance: $WCSensorLatency = KeypadSensorResource.latency + KeypadReader.WCET + KeypadStateMessage.period$

Using this formula we can calculate the worst-case latency of the sensor component. If we define the same for the other components, we can get the following: $WCSensorLatency + WCControllerLatency + WCActuatorLatency < 150ms$

The developer can formulate various expressions using these technique that contain the specific QoS requirements for the system. The fulfillment of these has to be assured by the development process, or development tools.

The platform independent model constructed here contains information about the functional blocks of the system, the communication of the blocks and also about the simple and complex QoS requirements. The next step is to map the elements to the implementation platform.

4 Implementation Models

After the analysis and functional design phase the following step of the system development is the selection of the appropriate implementation platform and the creation of platform specific model for the application. In most cases the platform specific model serves as a basis for automatic code generation.

4.1 Modeling platform components

In order to be able to complete the PIM to PSM mapping the MDA approach suggests using a so-called *platform description* that specifies the target platform. The format or content of this artifact is not further specified. We suggest using UML for platform modeling, as described in this section.

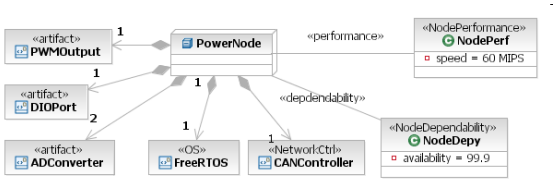

Fig. 4. Definition of a computing node, peripherals, and middleware

Figure 4 illustrates the definition of a computing node, its peripherals (digital I/O, A/D converters, and pulse-width modulated output) and the software environment (the operating system of embedded computers is like the middleware on application servers). Of course the elements have several parameters (memory size, processor architecture, and so on) that are not modeled in this small example. The performance and dependability related properties are grouped into separate stereotyped classes as in case of software components.

4.2 Platform to Application bindings

The model on Fig. 4 defines only the possible node type, while the actual hardware configuration can contain multiple instances of nodes connected together. This can be modeled using UML node instances and communication links. Upper part of Fig. 5 illustrates the instantiation of the nodes and the creation of a three node cluster. Only the relevant peripherals are shown in the figure.

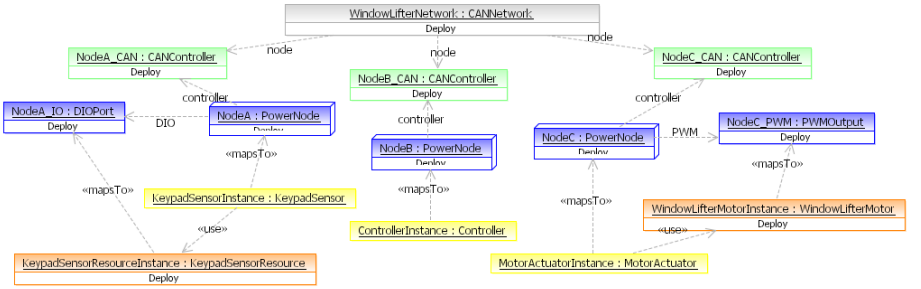

Fig. 5. Deployment of the example on a three-node cluster

The deployment of the system is defined by the mapping of component instances to node instances and logical resources to peripheral units. This is illustrated by the lower part of Figure 5. The deployment diagram is the basis of all configuration generation and analysis tools, because it shows the connection between hardware and software components.

5 Analysis of Implementation-level Models

After the platform specific model - that also includes the non-functional properties of the platform and the requirements of the application - of the application is completed, all information is present that is needed for QoS analysis. As stated in the introduction, we focus on performance and availability analysis in this paper, but the methodology can be applied for all other QoS aspects as well.

5.1 Performance Analysis

Performance analysis in this case calculates the worst case execution time of the application logic from the sensor state change to the actuator. This is an important information for the developer, because this is the only user-observable performance attribute of the system. We have to note also, that as a results all of the worst-case timings of component code execution and communication latency will also be calculated.

As analysis model we use the queueing networks that are widely used in telecommunication and queueing theory applications. The network consist of processing nodes that have input queues. The length of the queues and the processing time can be defined. The nodes are connected with connections that represent request flow paths.

We defined a transformation that maps the PSM of the application to a queueing network. The transformation builds the queueing model starting from the sensors. Before a sensor, a *source node* is inserted that generates inputs (request) for the model. The sensor is a simple delay that simulates the sensor delay of the actual hardware.

The data flows from the sensor to a job in the application. The jobs are running on the nodes as defined in the deployment plan. Nodes are modeled as *server node*s, and the processing time is set to the WCET of the job running on the node.

The inter-job communication channel is also modeled with a server node, where the queue length is 1, and the processing time is set according to the network parameters. Finally, actuators are mapped to a delay node that simulates the actuator latency and a sink node that is the final node of the network.

If multiple data paths are present (multiple jobs are mapped to a node) we can use different request classes that use the same server node but with other service times. The same is true for the communication channels. If a job sends multiple messages we insert a fork node after the job that produces a new token for all messages.

For evaluation of queueing networks we use the toolset developed at Politecnico di Milano [19], [5]. This toolset supports the definition, simulation, and analysis of queueing networks and the definition of timing parameters as probability variables. In this example we want to calculate the worst-case behavior of the system; therefore the transformation will set the parameters as constants for the worst-case values.

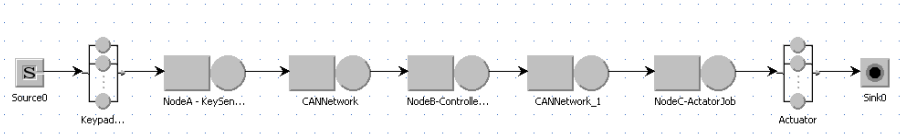

Fig. 6. Queueing network model of the sample application

Figure 6 shows the queueing model of the example system (note that the motor feedback is not shown here). As the CAN (Controller Area Network [7]) network is represented as two independent server node, the message transfer time can be set independent for the two messages (that is needed because CAN arbitration is priority-based and the two messages have different priority). These time parameters are calculated by a CAN schedule-checker application. More academic and industrial tools exists for this purpose (for instance [17]). The other parameters are directly presented in the model.

The UML to queueing networks transformation introduced here is a generic solution that can be found in the literature, but our added value is the integration of platform-specific tools (like the CAN schedule-checker) to automatically calculate performance related values and parameters for the target model.

The analysis framework calculates the response time of the system, but it can also be used for calculating response times for components. The analysis options can be customized. We must note that although the analysis tool is based on a mathematical notation, the designer does not have to use it directly, only if he wants to make custom analysis. Our development environment hides the analysis tools as far as only the predefined measures have to be calculated.

5.2 Dependability Analysis

The goal of the dependability analysis is to calculate the dependability-related properties of the system. We have chosen availability, one of this measures, as a demonstration example, but the other measures can also be calculated using the same concepts.

Availability in this context means the availability of the *service* that is provided by the application. The service in this case is the window lifting. We can say that the service is available if the actuator can execute our orders. Before the analysis the user has to select which output of the system is the service under evaluation.

The basic idea of the analysis model for availability is the mapping of the application model to an availability tree. The nodes of the tree are the hardware and software components, and the edges are the dependencies between them. A software node depends on a hardware node if it is deployed on that, and it depends on an other software node if it receives input from that. We assume that the hardware nodes are independent, so they do not have connections.

In the next step we assign availability values to nodes, if possible: hardware nodes have pre-defined availability (see Fig. 5) that can be the specification of the manufacturer, or result of a test. In our model, the hardware and the middleware (operating system) is a single unit.

We have a second assumption that the application software is free from coding errors. This can be true, because (i) most of the code is automatically generated, (ii) in critical systems the models and the code must undergo on rigorous verification and validation. This means that errors source in the hardware and the middleware.

After we have defined availability for hardware nodes, we can define the propagation rule for availability. For node number j, the availability can be calculated the following way:

$$\prod_{i \in \{x|(j,x) \in Edges\}} (availability(i)) \quad (1)$$

Using this formula we can propagate the availability values along the dependencies. Please note that this simple production can only be used if the hardware components are independent. The service availability value will be the value contained by the actuator node.

6 Implementation of the system

The approach described in the earlier sections uses several COTS components (UML modeling tool, analysis tools) that have to be integrated in order to achieve the interoperability of them. We have implemented a demonstration system in the framework of project DECOS [4] that integrates these tools and performs the model analysis of dependable embedded system models.

This framework receives the platform independent models and platform descriptions from UML tools or domain-specific editors and allows the definition of software-hardware integration. This system uses the VIATRA2 framework for transformation-based model integration.

We extended this model integration framework with the UML to analysis transformations that can generate the input for performance and availability modeling tools introduced earlier. This allows the transparent integration of these tools on the same platform.

Our experiences show that the integration can be achieved using the proposed method and the QoS analysis can be integrated to the development workflow. The system designer can work in a single environment and the analysis tools are invoked using automatic techniques.

7 Related Work

Application model analysis has been investigated by several researchers in the past. Foster et al. [8], [9] have worked out a method for verification of web service compositions, but only from the functional modeling viewpoint. Using that technique together with our solution is a combination for functional and non-functional analysis of service-based system models.

A survey on the possible model-based performance prediction methods has been done by Balsamo et al [21]. This work summarizes the most important methods in this area. However, it only contains performance evaluation without considering other aspects of non-functional properties.

Jin et al. [18] present a method for simulation-based analysis of web service compositions. They have focused on the timeliness analysis of applications, and do not calculate dependability related properties.

Current business process modeling tools that support the modeling of service composition like IBM Websphere Business Integration Modeler [3] are able to simulate the execution of service compositions but their capabilities are limited to the temporal domain. Our approach can also be used to complement the analysis capabilities with dependability analysis.

Similarly, the SCADE tool suite [22] can do model-checking of the behavioral model of the application, but is also limited to functional property verification. However, the source code generated by this tool is proven to be correct, the properties of the deployed system cannot be calculated by the tool.

There are several research results in this field that focus only on the derivation of analysis models and do not deal with the rest of the development workflow, like [11], [10]. These works are useful if we want to integrate new aspects to the analysis framework. The transformation of system models to analysis models can be created based on these results.

8 Conclusions

In the current paper we presented an approach for QoS-aware model-driven development of distributed applications. The development workflow includes the modeling and analysis of both software and hardware components. The results can be applied to multiple domains, for instance in case of distributed embedded systems and service-oriented business applications.

Currently we are successfully using this approach in European Integrated Project DECOS [4] for embedded systems, and have also experiments in the business application domain [20].

The QoS-aware development of these systems allows the system designers to investigate the non-functional system properties in the early phases of the development to avoid the performance and dependability problems in the testing an operation phase.

References

1. The W3C Consortium. Web service description language specification 1.1. http://www.w3.org/TR/wsdl.
2. The W3C Consortium. Web service level aggreements language specification 1.0. http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf, January 2003.
3. IBM Corporation. Ibm websphere business integration modeler tool homepage. http://www.ibm.com/software/integration/wbimodeler/.
4. DECOS. Dependable components and systems. an eu framework 6 integrated project. http://www.decos.at/.
5. Performance Evaluation Lab Dipartimento di Elettronica e Informazione Politecnico di Milano Italy. Java modelling tools project homepage. http://jmt.sourceforge.net/.
6. T. Andrews et al. Business process language for web services specification 1.1. ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf.

7. International Organization for Standardization (ISO). Controller area network specification 2.0. ISO Standards 11898-1 -2 -3, 2003.
8. Howard Foster, Sebastián Uchitel, Jeff Magee, and Jeff Kramer. Model-based verification of web service compositions. In *18th IEEE International Conference on Automated Software Engineering (ASE 2003), Montreal, Canada*, pages 152–163. IEEE Computer Society, 2003.
9. Howard Foster, Sebastián Uchitel, Jeff Magee, and Jeff Kramer. Tool support for model-based engineering of web service compositions. In *Proc. IEEE International Conference on Web Services (ICWS 2005), Orlando, FL, USA*, pages 95–102. IEEE Computer Society, 2005.
10. L. Gönczy. Dependability analysis of web service-based business processes by model transformations. In *Proc. of the First European Young Researchers Workshop on Service Oriented Computing (YR-SOC)*, pages 32–37, Leicester, UK, 2005.
11. L. Gönczy, A. Pataricza, F. Di Giandomenico, S. Chiaradonna, and A. Bondavalli. Dependability modeling of web service flows. In *In Supplementary Volume of the Fifth European Conference on Dependable Computing (EDCC-5)*, pages 77–78, Budapest, HUNGARY, 2005. Fast Abstract.
12. The Object Management Group. Model-driven architecture information portal. http://www.omg.org/mda.
13. The Object Management Group. Unified modeling language 2.0. http://www.omg.org/technology/documents/formal/uml.htm.
14. The Object Management Group. Uml profile for modeling quality of service and fault tolerance characteristics and metrics. http://www.omg.org/, September 2004.
15. The Object Management Group. Uml profile for schedulability, performance, and time specification. http://www.omg.org/, January 2005.
16. P. Peti H. Kopetz, R. Obermaisser and N. Suri. From a federated to an integrated architecture for real-time embedded systems. Technical report, Technische Universität Wien, Institut für Technische Informatik, 2004.
17. VECTOR Informatik. Canalyzer 5.2 product home page. http://www.vector-informatik.de/, 2006.
18. Akhil Sahai Li-jie Jin, Vijay Machiraju. Analysis on service level agreement of web services. Technical report, HP Laboratories Palo Alto, 2002. http://www.hpl.hp.com/techreports/2002/HPL-2002-180.pdf.
19. G.Serazzi M.Bertoli, G.Casale. Java modelling tools: an open source suite for queueing network modelling and workload analysis. In *Proceedings of QEST 2006 Conference*. IEEE Press, September 2006. In Press.
20. A. Pataricza, A. Balogh, and L. Gönczy. *Enterprise Modeling and Computing with UML*, chapter Verification and validation of non-functional aspects in enterprise modeling. Idea Group, 2006. In press.
21. P. Inverardi M. Simeoni S. Balsamo, A. di Marco. Model-based performance prediction in software development: a survey. *IEEE Trans. on Software Engineering*, Vol 30/5:pp. 295–310, May 2004.
22. Esterel Technologies. *SCADE Suite Technical and User Manuals, Version 5.0.1.* Esterel Technologies, June 2005.