Model Driven QoS Provisioning Architecture for Human Machine Interface Applications of In-vehicle Infotainment Systems

Hemant Sharma, Lokesh Madan and Dr. A. K. Ramani

Abstract—**In-vehicle infotainment applications often have to contend with different degrees of unpredictability. These applications can benefit if they can specify quality requirements to the service they access.**

This paper presents our attempt to apply Model Driven Architecture (MDA) technology to integrate the Quality of Service (QoS) requirements for In-vehicle Human Machine Interface (HMI) clients of infotainment applications. QoS subsystem of Human Machine Interface framework provides the essential QoS modelling support to specify QoS related requirements. QoS modelling constructs and components of the Infotainment Human Machine Interface Framework (iHMIFw) support construction of platform independent model of infotainment HMI applications with required QoS specifications. A model transformation has been described that is supported by the framework and results into XML description of QoS specifications.

Index Terms— **Human Machine Interface (HMI) Framework, Model Driven Architecture, Quality of Service, Unified modelling Language (UML).**

I. Introduction

The amount of infotainment software components in modern cars is increasing at a breathtaking pace. With the development of infotainment functionality, the car is becoming an information hub where functions of Mobile Phones, Laptops and PDAs are interconnected using wired and wireless network technologies (UMTS, Bluetooth, WiFi) via and with car information systems. Increasingly, the on-board electronics systems establish communication links beyond car boundaries, enabling various applications relating to, among others, sharing of infotainment content, remote vehicle analysis, software updates, global positioning and emergency services, inter-vehicle communication for crash prevention and automatic convoy forming.

Hemant Sharma is Software Engineer at Delphi Delco Electronics Europe GmbH, Bad Salzdetfurth, Germany. (e-mail: hemant.sharma @ delphi.com).

Lokesh Madan is Technical Manager at Delphi Delco Electronics Europe GmbH, Bad Salzdetfurth, Germany. (e-mail: hemant.sharma @ delphi.com).

Dr. A. K. Ramani, is Professor at School of Computer Science, Devi Ahilya University, Indore, INDIA. (e-mail: headscs@dauniv.ac.in).

In-vehicle infotainment applications often have to contend with different degrees of unpredictability, which arise from several factors, such as unanticipated faults in the system and transient overloads caused by sharing the available in-vehicle network and computing resources with other applications. Owing to this unpredictability, infotainment applications can benefit if they are allowed to specify and control the quality-of-service (QoS) they receive from the services they access. To support the diverse QoS requirements of next generation in-vehicle infotainment applications, we need to build QoS-aware HMI framework that allows the HMI clients to express their application-specific requirements using the right level of abstraction.

The concepts of platform independent and platform specific models (PIMs and PSMs) are fundamental in MDA [1, 2]. The PIM and PSM models currently tend to focus on how systems should realize the desired functionality. However, systems have additional properties that characterize the system's ability to exist in different environments and adapt as its environment varies. These extra-functional properties, also called qualities or quality of service (QoS), address how well this functionality is (or should be) performed if it is realized. In other words, if an observable effect of the system doing something can be quantified (implying that there is more than 'done'/'not-done' effect of the behavior), one can describe the quality of that behavior.

A transition from, for instance, a PIM to a PSM covers transformation of the concepts from the PIM to the PSM, which is some what more than a pure meta-model based transformation. To be useful, the transition process also needs to take into account and utilize the actual target technology. This includes use of common patterns and use of standard mechanisms, e.g., to satisfy the required quality of services provided by the system. Therefore, we believe utilization of QoS-specifications when performing model transformation and code generation is the key to gain efficient and useful results.

In this paper, we present a framework which enables development and deployment of QoS aware HMI applications for In-vehicle Infotainment platforms. This Infotainment HMI Framework (iHMIFw) allows clients to access replicated services by requesting QoS along two dimensions: timeliness of response and consistency of delivered data. While our framework ensures that the consistency requirements are always met, the uncertainty in most distributed environments makes it hard to provide deterministic guarantees for meeting the timeliness requirements of applications. Hence, our approach provides probabilistic temporal guarantees [3].

The rest of the paper is organized as follows: Section 2 provides the background. Section 3 introduces the iHMIFw framework and its core components. In section 4, we provide a short description of QoS subsystem of iHMIFw. In section 5, the elements of QoS modelling for HMI applications are discussed. Section 6 presents iHMIFw concepts for QoS modelling. We present our conclusions in Section 7.

II. BACKGROUND

Over the last few years, some approaches have been proposed for software adaptation using dynamic change in application components. [12] Presents an approach for dynamic reconfiguration of component-based applications for the Microsoft .NET platform. [13] Provides a framework for enabling dynamic adaptation of applications executing in dynamic environments.

It has been shown that application-level QoS specifications of each service instance are available and collocated with the service instance. Several programming environment and specification languages have been proposed to allow application developers to provide such QoS specifications [4, 5]. There exist translators that can map the application level QoS specifications into the resource requirements (i.e., CPU, memory, network bandwidth/latency, etc.). Such a translation procedure can be performed based on two major approaches: (1) analytical translation; and (2) offline/online probing services, which have been addressed by a wealth of research work [6, 7].

In order to assure the correct and complete specification of QoS requirements for In-vehicle multimedia and infotainment applications, it is necessary to follow a valid formal model, as described in [8].

The Unified Modelling Language [10, 11] is a collection of semi-formal models for specifying, visualizing, constructing, and documenting models of technical systems and of software systems. It provides various diagram types allowing the description of different system viewpoints. Static and behavioral aspects, interactions among system components and implementation details are captured via UML diagrams. UML is very flexible and customizable, because of its extension mechanism. [9] Presents the adopted UML profile for Schedulability, Performance, and Time. Although this profile does not cover QoS in a broad sense, defining only QoScharacteristic and QoSvalue classes in their conceptual model, our UML profile for QoS is aligned with their approach. We also follow their lightweight approach to specify a QoS profile.

Our research work has also been influenced by Options Configuration Modelling Language (OCML) [14], which is a Model Driven Development (MDD) tool that simplifies the specification and validation of complex Distributed Real Time and Embedded (DRE) middleware and application configurations, and Benchmark Generation Modelling Language (BGML) [15], which is the MDD tool that synthesizes benchmarking test suites to analyze the QoS performance of OCML-configured DRE systems.

III. FRAMEWORK ARCHITECTURE

In this section, we present the architecture of kernel of iHMIFw framework. iHMIFw is intended to provide a flexible platform to develop HMI applications for diverse category of Infotainment systems.

The kernel of iHMIFw consists of essential components that are necessary for HMI applications to function as expected. These core components interact with each other and with external environment either in synchronous manner or in asynchronous manner. The core components that form the kernel of iHMIFw are as follows:

A. *Context Engine*

This subsystem provides support for specification of context information for HMI applications. The context information may contain rules from context repository.

B. *Dialog Engine*

Dialog Engine maintains the overall look and feel of HMI views. It is also responsible for deciding the 'view transition'.

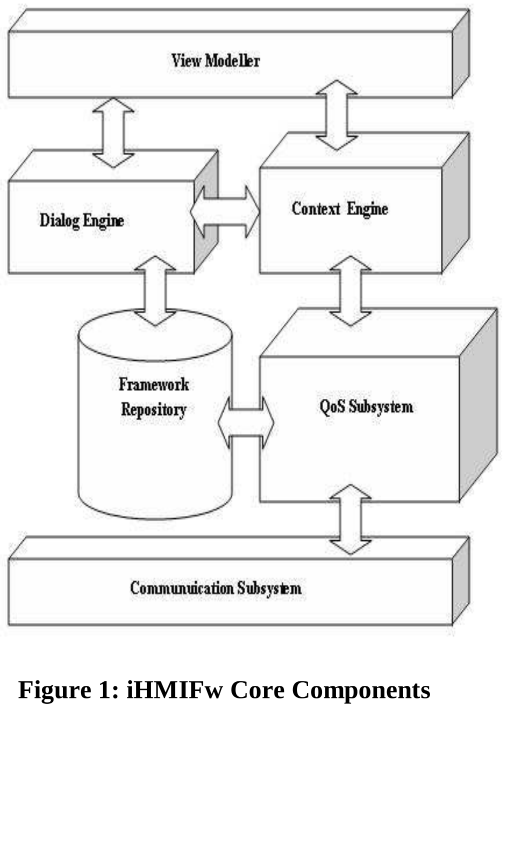

Figure 1: iHMIFw Core Components

C. *View Modeler*

This subsystem maintains the contents of a view and responsible for refreshing them.

D. *Widget Library*

This is the library of widgets that are used for creating the HMI views. This library is part of framework repository.

E. *Communication Subsystem.*

This subsystem provides an abstraction layer for inter-vehicle and external communication.

F. *QoS Subsystem*

QoS subsystem is responsible for specification and monitoring of QoS for HMI applications. This subsystem actively communicates with HMI applications for all QoS related issues.

IV. Framework QoS Subsystem

One of the important responsibilities of our QoS-aware iHMIFw is the selection of appropriate replicas to service the HMI client applications to meet their QoS requirements. One approach would be to select all the available replicas to service a single client. However, such an approach is not scalable, as it increases the load on all the replicas and results in higher response times for the remaining clients [3]. QoS specific features of iHMIFw are supported by QoS subsystem. This subsystem is collaboration of following components:

- *QoS Controller Factory,*
- *QoS Event Handler,*
- *QoS Rule Parser,*
- *View QoS Configurator.*

The QoS controllers provide different QoS management functions (e.g., (re)configuration, adaptation, scheduling) and may invoke the QoS-aware resource management functions (e.g., resource reservations). The QoS Event Handler keeps track of different view context information and triggers the appropriate QoS controller(s), provided by *QoS Controller Factory*, into action. The *QoS Rule Parser* translates the HMI application specific policies into desired data structures with help of the *QoS Event Handler*, which steers the QoS management towards the satisfaction of application-specific QoS criteria. The *View QoS Configurator* automatically derives the user's preferences and life routines, based on the user's inputs and QoS configuration library. These user inputs form the basis to personalize the QoS management.

QoS Event Handler is distributive in nature. To handle HMI client specific subscription of QoS events, it provides client side *Facade Proxy* interface. This facade proxy has corresponding *QoS Event Facade* at QoS subsystem of iHMIFw.

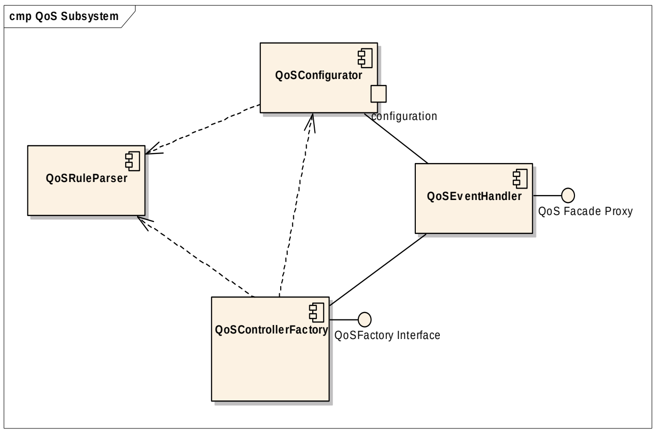

Figure 2: QoS Subsystem Architecture

The distributed nature of *QoS Event Handler* provides following advantages:

- it can handle much richer context information than the passive, stand alone QoS monitors;
- it promotes the separation of concerns to relieve the QoS controllers from the burden of handling complex events;
- it enables loose coupling between entities to ease the development and management of ubiquitous infotainment applications;
- It allows the QoS management functions to be changed and personalized dynamically during runtime.

V. QoS Modelling Elements

In this section, we present the building blocks that are used to construct PIM for QoS enabled HMI applications.

A. *Architectural Pattern*

The HMI applications based on iHMIFw has following characteristics:

- These applications use iHMIFw subsystems via standardized functional interfaces.
- The quality of the services provided by iHMIFw subsystems is critical to the application's conformance to its requirements.
- The application's QoS requirements maps directly to the QoS categories defined or configured at QoS subsystem of iHMIFw.

It is required that the quality sensitive HMI applications should be able to specify, monitor and control the QoS of its supporting components.

QoS Subsystem implements a stripped down *Quality Connector* pattern for each framework infrastructure subsystem that provides only a non-standard QoS-control interface. The objects of *Quality Connector* pattern configure the iHMIFw subsystems to provide, if possible, the requested QoS in the specified modes of operation of HMI application.

The *Quality Connector Pattern*, used here, includes following participants.

Static QoS Configurator: This object scans the *View Tree* description of HMI application for QoS related statements and declarations. It then provides these statements to QoS subsystem to generate corresponding target specifications.

Run-Time Connector: This acts as a plug-in for HMI applications at runtime. When the quality connector object receives a request for a QoS contract, it uses the *Configuration object* or similar mechanism to discover the infrastructure base-objects and corresponding meta-objects that might be used to provide the requested service in the specified mode.

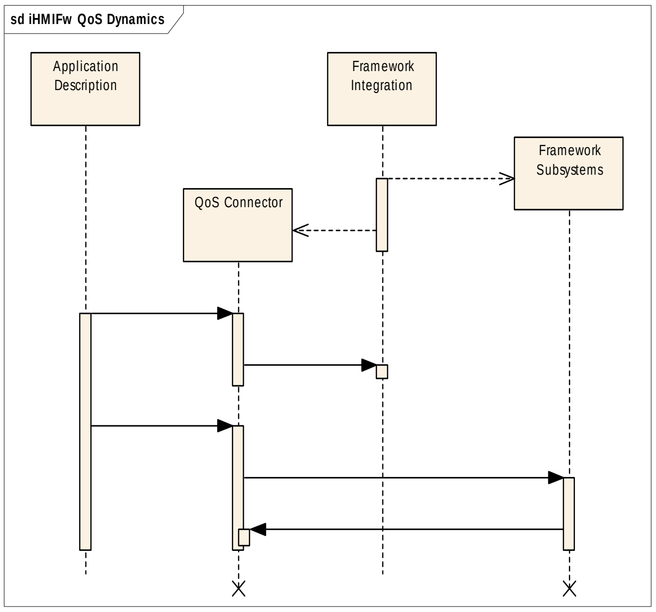

Figure 3: iHMIFw QoS Dynamics

Figure 3 provides the dynamics of *Quality Connector* pattern in context of an HMI application.

When the components of the iHMIFw subsystems, to which QoS requests will be made, are known, the HMI dialog description is, if necessary, modified automatically to insert the statements that make the runtime requests. Framework infrastructure components are selected and constructed using whatever information is known about the required QoS and peak load imposed on the target service. At runtime, the HMI application requests a QoS in a specified mode, including the specification of work load. The *Quality Connector* pattern object determines whether the request could be satisfied using the available functionality of framework subsystem, considering the QoS contracts accepted previously. If the request is found to be feasible, the application is granted a contract, and the strategy by which the service would be provided is recorded. Further, QoS listeners are attached to the configuration items whose mode changes might signal transition to or from the relevant mode.

B. *QoS Profile*

In this section, we present the customized UML profile for QoS that is being used by iHMIFw. From the domain viewpoint, we define the QoS modelling elements needed to specify QoS of a component (or object) that provides a service subjected to QoS constraints, and to associate these QoS specifications with the component.

The iHMIFw introduces a stereotype called *QoSContract* that extends the *Class* metaclass of UML [10]. This introduction is based on the fact that the HMI contract types can be instantiated almost like classes in object oriented languages. A *View Description* becomes a stereotyped *UML Attribute*, which is pretty straightforward because a class contains attributes. Similarly, *QoSContract* shall contain one or multiple instances of QoSViewDescription. The attributes of metaclass *QoSViewDescription* shall become so called *tagged values* in the stereotype.

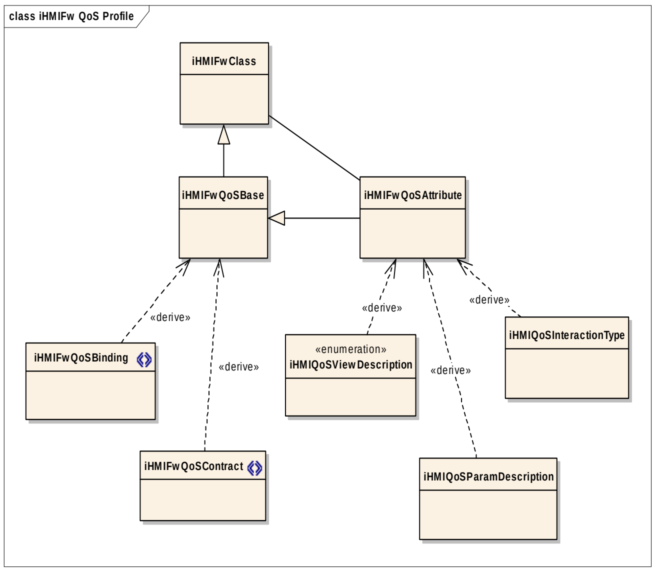

Figure 4: Extended QoS profile

A *QoSParamDescription* is considered to be a stereotyped class, too. Therefore, a *QoSParamDescription* is mapped to an attribute just like *QoSInteractionType*. This makes it clear that a contract does not directly contain a *QoSParamDescription*. Instead it can have a normal UML attribute which has a type that in turn presents a *QoSParamDescription*. This seems to introduce some structural differences between iHMIFw QoS meta-model and the UML profile. However, we map concepts from UML profile one to one so that we can translate the UML model in an extended QoS configuration file without loss.

In UML, components provide interface through which they can be connected by a *Realization* dependency. Instead of extending the UML metaclass *Realization*, we have introduced *QoSBinding*, so that we can map our QoS metaclass. The UML components always play the role of context for *Realization dependency*. In iHMIFw, the connection between component *QoSContract* interface and the *QoSBinding* is depicted by a derivation. Further, the connection between *binding* and *contract* may also be depicted by a normal dependency (dashed line with open arrow). If there are no additional dependencies present then the contract is assumed to be negotiable for all interfaces of the component at once. Otherwise iHMIFw draws a normal dependency between the *QoSBinding* and one or more interfaces of component to indicate that the *QoSContract* only affects these interfaces. An interaction type, represented by stereotype QoSInteractionType, is needed if the contract is suppose to affect only certain methods of an interface. *QoSInteractionType* is a UML Attribute and therefore it can store a reference to some UML Operation. This stereotyped attribute is owned by the class, which is stereotyped as QoSBinding. Figure 4 gives an overview of QoS profile for iHMIFw.

VI. QoS Modelling

In this section, we describe our approach for platform independent modelling of QoS for HMI applications. The model transformation approach, presented in the later subsection, provides an overview for generating an XML based QoS model.

A. *Platform Independent Modelling*

Modelling QoS-enabled HMI applications in a platform independent manner require a platform independent modelling language (PIQML) enriched with QoS concepts. The QoS subsystem of HMI framework has adopted UML to provide abstract syntax, rules, notations and semantics for QoS specification. The subsystem uses customized meta-model which support QoS categories. It is possible to refer the QoS categories by name. The meta-model also provides means for modelling constraints upon these QoS categories. In order to express the semantics of QoS categories in the model the semantics have been mapped to model elements that are well understood and non-ambiguous. The semantics of QoS category are further linked to a set of measurement parameters and evaluation functions. The QoS subsystem shall maintain repository of measurement parameters and evaluation function for mapping of QoS categories.

iHMIFw uses component based approach for design of HMI applications, in order to have strict encapsulation. The components shall use defined ports to establish communication with other components or with external environment as well as with the framework. The advantage of this approach is that it separates QoS-related modelling elements from the business logic. The business logic of HMIs shall be encapsulated inside the components based on iHMIFw framework, whereas the QoS properties shall be attached to the ports of these components.

Figure 5 partly presents PIM development approach for HMI applications. Here, the HMI QoS specifications are separated into Appearance *Contract* and *Behavior Contract*. Explanation of these contracts is beyond the scope of this paper.

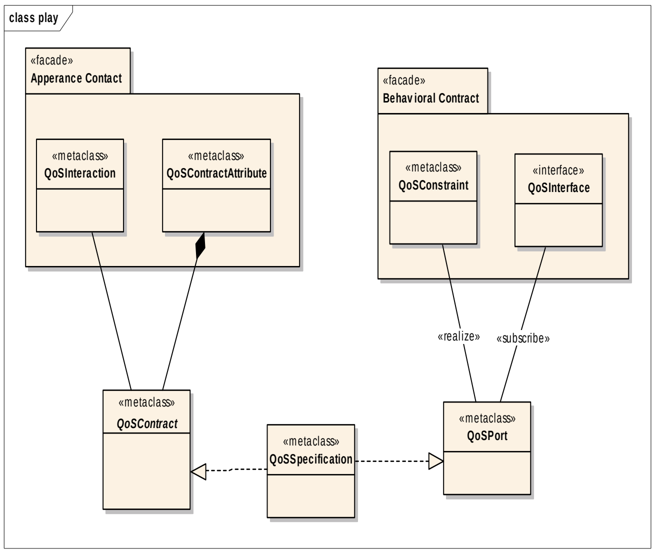

Figure 5: Platform Independent QoS Modelling

B. *QoS Transformations*

Key challenges, for implementing the model-driven paradigm to QoS specification modeling in iHMIFw, are to define, manage, and maintain traces and relationships between the platform independent models and independent model views. This again, is the foundation for performing model transformation and code generation. Therefore, transforming a QoS PIM model to PSM model is done by means of a generic transformation specification that specifies how a meta-model description of the source model should appear in the target model.

The iHMIFw extends the *XML QoS specification language* (XQoS) [16] for transformation of PIM for QoS. The extended XQoS uses following main elements to frame the QoS specifications:

Synchronization and timing elements: The tags <sync> and <prac> are intended to describe the inter-flow data synchronization constraints for HMI applications. These elements shall contain additional tags to detail the constraints to describe the *QoS contract* in finer way.

Flow Description Elements: These elements intend to identify the type constraint on data flow tags <route data>, <control message> and <view element>.

View Transition Elements: These elements preset constraints on HMI view transition and view transition information. The tags <condition>, <immediate> and <composite> describe constraints on view transition.

Figure 6 below, presents the extended XQoS transformed model for *route data request* for showing *Route Map* for turn-by-turn guidance.

```
<Contract>
  <DATAFlow>
    <!--- extended XQoS specification for Route Data Flow --- >

      <SYNC repeat="loop" maxTimeOut="10ms"
              minReliability="80%" >
        <!-- Route Data sequence with 20% losses permitted >

          <DATAType command="route data request" >

            <CONSTRAINTS packetsize="4096" maxPackets
                                        ="1024">
            </CONSTRAINTS>
          </DataType>
      </SYNC>
    </DATAFlow>
</Contract>
```

Figure 6: Extended XQoS Mapping

Extensions to XQoS are based on a formal specification model which is sufficient to describe completely and rigorously, the QoS needs of existing and new infotainment applications. Application of extended XQoS constructs to an experimental *Navigation HMI* prototype, for *intra* and *inter-flow* QoS description, resulted in adequate QoS requirement description in form of XML files of optimal size. Details of the specification model and the prototype are out of scope of this paper. The experiences with the prototype encourage us to claim that the QoS description using extended XQoS will provide optimal transformation of PIM for HMI to a PSM XML description.

VII. Conclusion and Future Work

Emerging In-vehicle Infotainment and Telematics applications bring new challenges to the automotive infotainment and telematics platform research. In this paper, we have presented a novel QoS provisioning architecture to address the QoS challenges in the area of infotainment HMI.

We have presented an approach to specify QoS requirements of infotainment HMI applications with UML designs using MDA technology. For this purpose, extensions from the UML Profile for Schedulability, Performance, and Time have been used to introduce the necessary additional information into *Platform Independent* UML Model. A method for the transformation of the resulting UML model into corresponding extended XQoS [16] specifications has been proposed.

This contribution introduces the *QoS subsystem* and QoS elements that has been integrated to iHMIFw. Our work identifies a standard means to specify application level QoS requirements for infotainment HMI applications at design time.

We propose to continue our work on iHMIFw to support the development of so-called 'Dynamic Service HMI applications' and specification of their QoS requirements. QoS subsystem shall be extended for dynamic QoS adoption and enriched set of extended XQoS constructs.

References

[1] Object Management Group. Model Driven Architecture (MDA), OMG Document ormsc/2001-07-01 edition, July 2001.
[2] D.S. Frankel, "Model Driven Architecture – Applying MDA to Enterprise Computing", OMG Press, Wiley Publishing, 2003
[3] S. Krishnamurthy, W. H. Sanders, and M. Cukier. A Dynamic Replica Selection Algorithm for Tolerating Timing Faults. In Proc. of the International Conference on Dependable Systems and Networks, pages 107–116, July 2001.
[4] X. Gu, K. Nahrstedt, W. Yuan, D. Wichadakul, and D. Xu. An XML-based Quality of Service Enabling Language for the Web. Journal of Visual Language and Computing, Special Issue on Multimedia Language for *the Web*, 2002.
[5] Joseph P.Loyall, David E. Bakken, Richard E.Schantz, John A.Zinky, David A.karr, Rodrigo Vanegas, and Kenneth R.Anderson. QoS Aspect Languages and Their Runtime Integration. Lecture Notes in Computer Science, 1511, May 1998.
[6] Tarek F. Abdelzaher. An Antomated Profiling Subsystem for QoS-Aware Services. Proc. of IEEE Real-Time Technology and Applications Symposium, June 2000.
[7] B. Li and K. Nahrstedt. QualProbes: Middleware QoS Profiling Services for Configuring Adaptive Applications. Proc. of IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware 2000), April 2000.
[8] Synchronized Multimedia Group of the W3C Synchronized Multimedia Integration Language (SMIL) 3.0, Dec., 2006.
[9] Object Management Group. UML profile for schedulability, performance, and time. www.uml.org, March 2002.
[10] Object Management Group. Unified Modelling Language Specification v.2.0. www.uml.org, September 2003.
[11] Bondavalli, A., Dal Cin, M., Latella, D., Majzik, I., Pataricza, A., Savoia, G.: Dependability analysis in the early phases of UML-based system design. International Journal of Computer Systems Science & Engineering 16(5) (2001) pp. 265-275
[12] A. Rasche and A. Polze. Configuration and Dynamic Reconfiguration of Component-based Applications with Microsoft .NET. Proc. of 6th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, 2003.
[13] A. Mukhija and M. Glinz. A Framework for Dynamically Adaptive Applications in a Self-organized Mobile Network Environment. Proc. of ICDCS 2004 Workshop on Distributed Auto-adaptive and Reconfigurable Systems, 2004.
[14] E. Turkay, A. Gokhale, and B. Natarajan. Addressing the Middleware Configuration Challenges using Model-based Techniques. In Proceedings of the 42nd Annual Southeast Conference, Huntsville, AL, Apr. 2004. ACM.
[15] A. S. Krishna, D. C. Schmidt, A. Porter, A. Memon, and D. Sevilla-Ruiz. Improving the Quality of Performance-intensive Software via Model-integrated Distributed Continuous Quality Assurance. In Proceedings of the 8th International Conference on Software Reuse, Madrid, Spain, July 2004. ACM/IEEE.
[16] Ernesto Exposito, Mathieu Gineste, Romain Peyrichou, Patrick Sénac, Michel Diaz, Serge Fdida. XML QoS specification language for enhancing communication services, Proceedings of the 15th international conference on Computer communication, 2002, pp 76- 90.