

# Model-Based Design of Runtime Adaptation Strategies

Joseph P. Loyall<sup>1</sup>, Richard Shapiro<sup>1</sup>, Sandeep Neema<sup>2</sup>, Sherif Abdelwahed<sup>2</sup>, Richard Schantz<sup>1</sup>, Nagabhushan Mahadevan<sup>2</sup>

<sup>1</sup>BBN Technologies, Cambridge, Massachusetts

<sup>2</sup>Vanderbilt University, Nashville, Tennessee

## 1 Problem Description

Designing and implementing embedded computing systems is more challenging than non-embedded systems because of the need to interact closely with a changeable physical universe that operates in real time, and because of the extra constraints imposed by the packaging and environmental concerns for many embedded systems. In addition, the challenges posed by distributed systems architectures and composite implementations of stand-alone components (which are already prevalent in non-embedded environments) have begun to become commonplace consequences of the requirements for new embedded solutions. There are especially challenging issues surrounding the design and implementation of distributed embedded systems with managed quality of service (QoS) properties when those systems must adapt to the changes in both the computational and physical universes within which these systems must operate.

To date, research has concentrated on constructing runtime system abstractions and mechanisms that can adaptively meet the design requirements. However, using these abstractions and mechanisms requires highly skilled individuals with strong intuitions about both the needs of the domain and the ways to manipulate the various dimensions contributing to the managed QoS behavior. There are few design paradigms to guide the design or construction of this sort of dynamic, adaptive, managed runtime behavior. There is a compelling need for applying design-time methodologies to develop and control these runtime adaptations systematically, and along the way to establish appropriate interchanges and interfaces between the design-time tools and their runtime counterparts.

Under the auspices of the DARPA MoBIES program, we are investigating the application of model integrated computing (MIC) to the problems of designing, customizing, parameterizing, and managing the adaptive runtime characteristics of distributed realtime embedded (DRE) applications. In particular, we are seeking to utilize and augment the Generic Modeling Environment (GME) MIC tool to design runtime control capabilities as provided by the Quality Objects (QuO) adaptive QoS middleware framework. GME uses domain models and advanced user/designer interfaces to manipulate elements in the domain space that are intelligently interpreted by the models to produce effective designs and code elements representing that design [4]. QuO provides a set of extensions to existing off-the-shelf middleware, including CORBA and Java RMI, specification languages, and a runtime system to support QoS awareness and management of, and adaptation to, dynamic conditions [18].

---

This work is sponsored by the DARPA/IXO Model-Based Integration of Embedded Software program, under contract F33615-02-C-4037 with the Air Force Research Laboratory Information Directorate, Wright Patterson Air Force Base.

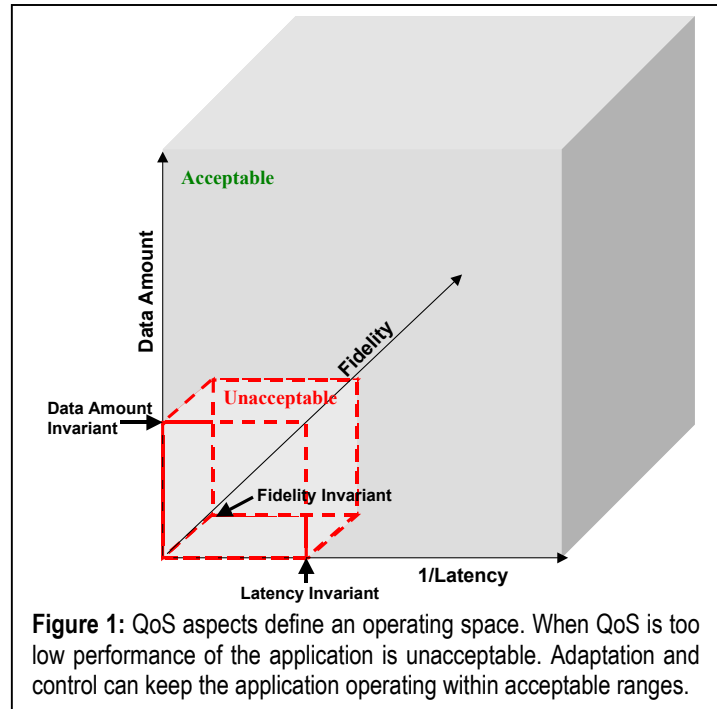
## 2 Approach to Modeling Runtime Adaptive Behavior

Our approach is in developing a semantically rich, domain-specific modeling language supporting the high-level design/representation of QoS adaptation strategies and concerns. We are designing and implementing mappings and code generators to derive runtime interfaces and middleware constructs from the high-level representation. We plan to link the design-time modeling tool and runtime middleware using a feedback loop for incorporating information gathered at runtime to feed back into refinement of the design time model.

The first step in our technical approach is to identify the relevant runtime QoS characteristics of DRE systems suitable for design-time modeling. The QoS aspects define a space, illustrated in Figure 1, in which the application has unacceptable quality of service attributes if the measure of any dimension of interest falls below the minimal operating threshold. Above a maximum operating threshold, improvements in QoS make no difference in application operation. Between the minimum and maximum thresholds is a space in which tradeoffs and adaptations can be made in order to maintain as high a level as feasible of acceptable quality of service. While Figure 1 illustrates three dimensions – corresponding to the dimensions of amount of data, fidelity, and latency – the space can consist of any number of dimensions, 1, 2, ... N, corresponding to the QoS dimensions relevant to the application.

While the minimum and maximum thresholds define an acceptable operating space, it is usually not the case that every point in the acceptable space is equivalent. Nor is it true that it is always possible to move smoothly from one point in the space to another. The underlying system, mechanisms, and resource managers provide *knobs* to control the level of QoS (e.g., through resource allocation, rate or priority adjustment, or data shaping). The granularity at which QoS can be adapted depends on these knobs. Furthermore, adaptation of one QoS dimension will frequently affect other QoS dimensions (e.g., increasing the amount of data will use more bandwidth). Finally, the application's requirements will often mean that some tradeoffs and adaptations are preferred over others. The main goal of our research is modeling *Adaptation Strategy Preferences*, which indicate how the adaptation moves the application through the acceptable operating space as system, functional, and mission conditions change. These preferences specify which adaptation behaviors and mechanisms are employed, the manner and order in which they are employed, the tradeoffs to be made, and the conditions that trigger adaptation. To this end, our modeling language supports modeling the following components of a QoS adaptive runtime system:

- *The application's structure* – Primarily the functional structure of the application, including data and control flow and points for inserting adaptive decisions.
- *Mission requirements* – The functional and QoS goals that must be met by the application. These help determine the relative merit of possible adaptations and points in the adaptation space.



- *Observable parameters* – The system conditions that need to be monitored at runtime in order to determine the system’s QoS state and drive adaptation. Examples include latency, throughput, and bandwidth between a pair of hosts, as well as reflective information about application execution, such as the nature of data content or the speed of operation. These parameters determine the application’s current position in the adaptation space.
- *Controllable parameters and adaptation behaviors* – The knobs available to the application for QoS control and adaptation. These can be in the form of interfaces to mechanisms, managers or resources, or in the form of packaged adaptations, such as those provided by QuO’s *Qosket* encapsulation capability [13].
- *The system dynamics* – The interactions between observable and controllable parameters and adaptation behaviors. These help define the set of possible trajectories that an application can take through the N-dimensional QoS parameter space.
- *Minimum and maximum acceptable ranges of operation* – This includes the lower bound on the level of acceptable QoS below which the system is unacceptable for a given mission and the upper bound above which additional resources lead to no improvement in system QoS.
- *Adaptation strategies (controller model)* - The adaptation strategy specifies the adaptations employed and the tradeoffs made in response to dynamic system conditions in order to maintain an acceptable mission posture.

We are using the graphical meta-modeling environment of GME [4] to define the modeling language. Our initial implementation builds upon ongoing efforts using GME for modeling the program’s functional structure graphically, using icons to represent program components and connections to represent control and data flow between application components. We model controllers as state machines and as difference/differential equations. Observable and controllable parameters are atoms in GME, representing interfaces to system primitives, managers, mechanisms, and so forth. System dynamics, mission requirements, and ranges of operation are expected to be functions defined on observable and controllable parameters.

In our current implementation concept, application structure maps to functional specification of the program’s structure, such as CORBA IDL descriptions, control flow, and dataflow. Observable and controllable parameters map to QuO system condition objects, which provide interfaces to mechanisms and managers. QuO has an increasing library of qoskets, encapsulated and reusable behaviors, which will serve the role of adaptation behaviors. We are exploring how to map mission requirements, system dynamics, and minimum and maximum acceptable ranges of operation.

One of the key components of our high-level modeling is specifying the adaptation strategy. We are approaching it, in part, from a control theory point of view. We plan a hierarchy of controllers, the outermost of which tries to maximize a measure of *utility* computed from the observable and controllable parameters in a manner defined by the mission requirements and system dynamics. We have identified and defined two types of controllers that we expect to be useful for modeling controlled behavior and generating predictable QoS adaptive middleware software: *supervisory controllers* and *classical compensators*. Supervisory controllers are represented by finite state machines, readily modeled in GME and then translated into QuO contracts [8], the QuO construct for specifying, controlling, and negotiating QoS adaptation [5]. We will model classical compensators as difference/ differential equations, from which native language functions, contained in qoskets and referenced by contracts, will be generated.

Figure 2 illustrates an example controller that we have modeled and simulated in our first prototype development of these concepts. It implements an adaptive strategy for a signal processing application based on a measure of *utility*. We use two observable/controllable parameters, *queue length* and *confidence* of the signal analysis. Queue length is representative of the signal latency, indicating how rapidly the application is processing signals compared to the rate at which they are coming in, i.e., the queue length will grow

if the signal analysis is processing signals slower than the rate at which they are appearing and will shrink (or stay stable) if processing is faster than the incoming rate. Confidence is a value provided by the signal analyzer based upon the amount of signal data it has processed and the amount of processing it has performed on the data. In general, the signal analyzer will have more confidence in its conclusions when it has processed a larger window of signal data and/or performed more feature extraction operations on the data. However, the increase in confidence is not a linear function. After processing a certain amount of data and extracting a certain number of features, more data and processing provide little, if any, increase in confidence. The utility function combines these two observable parameters, using weights assigned to each. The weights are provided by the mission requirements, which determine the relative importance to the application of processing more signals versus processing signals more accurately.

The modeled controller, realized in runtime by a QuO contract, tries to maintain the proper controlled balance of high confidence and low latency, by adjusting the queue length and processing characteristics. At each step of signal analysis, the controller contract decides whether the signal analyzer gets a chunk of data from the buffer (processing another piece of the current signal), thereby attempting to increase confidence in the analysis, or the next signal from the queue, thereby attempting to reduce latency. A more detailed discussion of this controller, the utility function, and the application can be found in [1].

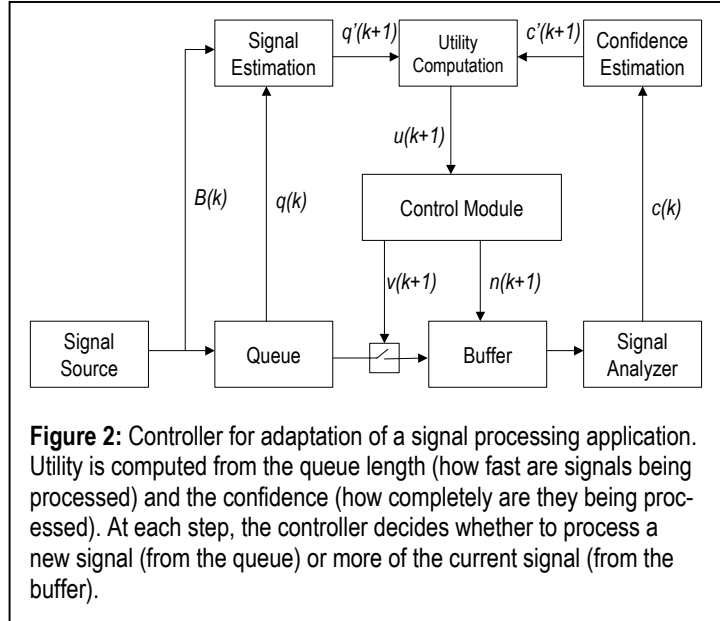
The modeled controller, realized in runtime by a QuO contract, tries to maintain the proper controlled balance of high confidence and low latency, by adjusting the queue length and processing characteristics. At each step of signal analysis, the controller contract decides whether the signal analyzer gets a chunk of data from the buffer (processing another piece of the current signal), thereby attempting to increase confidence in the analysis, or the next signal from the queue, thereby attempting to reduce latency. A more detailed discussion of this controller, the utility function, and the application can be found in [1].

### 3 Status of the Research and Ongoing Work

So far, we have developed the paradigm for modeling adaptive behaviors described above, and evaluated its utility in modeling a simple, adaptive signal analysis application. A screen shot of the adaptive signal analysis model is shown in Figure 3. We have simulated this application using MATLAB and are in the process of creating a runtime instantiation for it using the QuO middleware, while developing code generators for much of the QuO middleware aspect constructs needed to generate the runtime system [6]. We will be building on this by modeling and generating more capable adaptive DRE systems, using the signal analysis domain and our existing UAV and avionics applications [7] as examples, in order to extend, enhance, and evaluate our adaptation modeling language, code generation capabilities, and runtime middleware support.

### 4 Comparison with Other Work

There are a number of related efforts in the areas of QoS adaptive middleware and model integrated computing. Many of the QoS adaptive middleware efforts, such as ACE/TAO [14, 15], CIAO [17], RTCORBA [10], and FTCORBA [9], focus on distribution or infrastructure middleware and services that are complementary to our efforts. QuO works with these to provide a higher level, more dynamic policy and adaptation layer for combining and controlling the available mechanisms and services for end-to-end, dynamic solutions. Other research efforts focusing on QoS adaptive middleware, such as the University of



Illinois's Agilos middleware project, are pursuing similar goals and are building upon each other's advances.

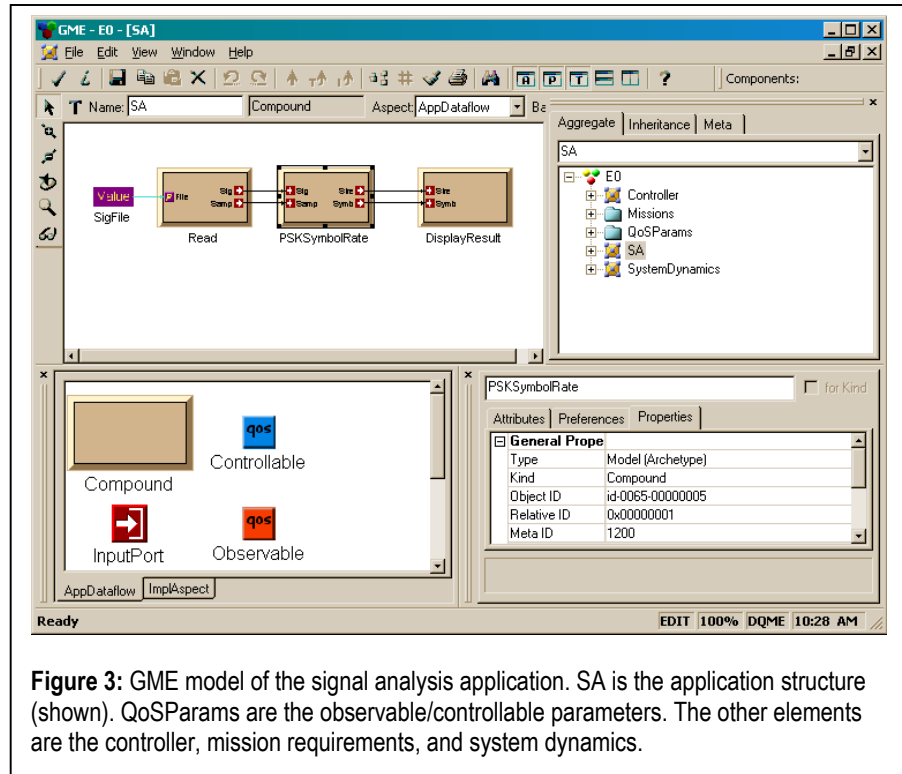
For modeling environments, there are some commercially available toolsets, such as Simulink and MATLAB, which we are using as part of our total solution. In addition, there are a few other meta-programmable tools available and emerging. Meta-Edit+, a meta-programmable tool from MetaCase, has capabilities for constructing domain-specific graphical languages built around the classical attributed entity-relationship concept. However, because it uses a relatively simple, proprietary reporting definition language to which only read access is provided, it is difficult to integrate Meta-Edit+ in a tool-chain. Honeywell's Dome is another meta-programmable tool with capabilities similar to Meta-Edit+ in being able to construct and instantiate domain-specific graphical languages and create modeling environments. The meta-programming environment of Dome is much more mature and powerful than that of Meta-Edit+ in that it has a graphical modeling environment instantiated within Dome itself.

Other research efforts have investigated the modeling of control-based QoS. For example, Sha et al also investigate an approach to modeling controllers as differential and difference equations in [16], including systems with highly variable workload. Fischer et al present a Petri Net approach to modeling QoS management systems [2]. There is also an effort within the OMG to standardize a UML approach for modeling QoS characteristics of systems [3, 11, 12].

Other research efforts have investigated the modeling of control-based QoS. For example, Sha et al also investigate an approach to modeling controllers as differential and difference equations in [16], including systems with highly variable workload. Fischer et al present a Petri Net approach to modeling QoS management systems [2]. There is also an effort within the OMG to standardize a UML approach for modeling QoS characteristics of systems [3, 11, 12].

## 5 Conclusions

We reported on ongoing work in the area of modeling adaptive behaviors and adaptation strategies and generating QoS adaptive middleware code for realizing them at runtime. Our work builds on the GME meta-modeling environment from Vanderbilt University and BBN's QuO QoS adaptive middleware. Our approach is to provide an environment for capturing an application's structure, operational requirements, minimum and maximum QoS thresholds, observable and controllable parameters, system dynamics, and adaptation control strategy. The model capturing these attributes would then synthesize the middleware code needed to add the described QoS awareness, control, and adaptation to the application. This work is ongoing, but our initial progress in describing our QoS adaptation modeling language and applying it to the area of signal analysis is promising.



**Figure 3:** GME model of the signal analysis application. SA is the application structure (shown). QoSParams are the observable/controllable parameters. The other elements are the controller, mission requirements, and system dynamics.

## References

- [1] S. Abdelwahed, S. Neema, N. Mahadevany, J. Loyall, R. Shapiro. "Online Hybrid Control Design for QoS Management," submitted to 42nd IEEE Conference on Decision and Control, December 2003, Maui, Hawaii.
- [2] S. Fischer, H. de Meer. "QoS Management: A Model-Based Approach," *Proceedings of MASCOTS'98*, IEEE Computer Society.
- [3] I-Logix, THALES, and Tri-Pacific. "UML profile for QoS and FT Characteristics and Mechanisms Joint Initial submission," OMG document number realtime/02-09-01.
- [4] A. Ledeczki, M. Maroti, A. Bakay, G. Karsai, J. Garrett, C. Thomason, G. Nordstrom, J. Sprinkle, P. Volgyesi. "The Generic Modeling Environment," WISP'2001, May 2001, Budapest, Hungary.
- [5] J. Loyall, P. Rubel, M. Atighetchi, R. Schantz, J. Zinky. "Emerging Patterns in Adaptive, Distributed Real-Time, Embedded Middleware," OOPSLA 2002 Workshop - Patterns in Distributed Real-time and Embedded Systems, November 2002, Seattle, Washington.
- [6] J. Loyall, D. Bakken, R. Schantz, J. Zinky, D. Karr, R. Vanegas, K. Anderson. "QoS Aspect Languages and Their Runtime Integration," *Lecture Notes in Computer Science*, Vol. 1511, Springer-Verlag, 1998.
- [7] J. Loyall, J. Gossett, C. Gill, R. Schantz, J. Zinky, P. Pal, R. Shapiro, C. Rodrigues, M. Atighetchi, D. Karr. "Comparing and Contrasting Adaptive Middleware Support in Wide-Area and Embedded Distributed Object Applications," 21st IEEE International Conference on Distributed Computing Systems (ICDCS-21), April 2001, Phoenix, AZ.
- [8] S. Neema, T. Bapty, J. Gray, A. Gokhale, "Generators for Synthesis of QoS Adaptation in Distributed Real-time Embedded Systems", ACM SIGPLAN/SIFSOFT Conference, GPCE 2002, Pittsburgh, PA, USA, October 2002.
- [9] Object Management Group. Fault tolerant CORBA. OMG Technical Committee Document formal/2001-09-29, September 2001.
- [10] Object Management Group. Real-time CORBA. OMG Technical Committee Document formal/2001-09-28, September 2001.
- [11] Object Management Group. "UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms, Request for Proposal," OMG Document: ad/2002-01-07.
- [12] Open-IT. "Response to the OMG RFP for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms," Document Version 1.0, OMG document number realtime/2002-09-02.
- [13] R. Schantz, J. Loyall, M. Atighetchi, P. Pal. "Packaging Quality of Service Control Behaviors for Re-use," ISORC 2002, The 5th IEEE International Symposium on Object-Oriented Real-time distributed Computing, April 29 - May 1, 2002, Washington, DC.
- [14] D. Schmidt. "The Adaptive Communication Environment: Object-Oriented Network Programming Components for Developing Client/Server Applications," 12<sup>th</sup> Annual Sun Users Group Conference, June 1994, San Francisco, CA.
- [15] D. Schmidt, D. Levine, S. Mungee. "The Design and Performance of the TAO Real-Time Object Request Broker", *Computer Communications Special Issue on Building Quality of Service into Distributed Systems*, 21(4), pp. 294—324, 1998.
- [16] L. Sha, X. Liu, Y. Lu, T. Abdelzaher. "Queueing Model Based Network Server Performance Control," *Proceedings of the 23<sup>rd</sup> IEEE Real-time Systems Symposium (RTSS '02)*.
- [17] N. Wang, D. Schmidt, A. Gokhale, C. Gill, B. Natarajan, C. Rodrigues, J. Loyall, R. Schantz. "Total Quality of Service Provisioning in Middleware and Applications," *Microprocessors and Microsystems spec. iss. on Middleware Solutions for QoS-enabled Multimedia Provisioning over the Internet*, 2003.
- [18] J. Zinky, D. Bakken, R. Schantz. "Architectural Support for Quality of Service for CORBA Objects," *Theory and Practice of Object Systems* 3(1), 1997.