

Model-Based Design of End-to-End Quality of Service in a Multi-UAV Surveillance and Target Tracking Application

Joseph P. Loyall¹, Jianming Ye¹, Sandeep Neema², Nagabhushan Mahadevan²

¹BBN Technologies, Cambridge, Massachusetts

²Vanderbilt University, Nashville, Tennessee

1 Introduction

Designing and implementing large distributed, real-time embedded (DRE) computing systems is challenging because these applications need to be able to react in real-time to changes in missions, conditions and operating environments within the scope of their domains. There are especially challenging issues surrounding the design and implementation of DRE systems with managed end-to-end quality of service (QoS) properties when those systems must adapt to the changes in both the computational and physical universes within which they operate [12].

While advances in design methodology especially model-integrated computing (MIC) [10] and middleware technologies [7,11,13] have made designing and developing the functional logic of DRE systems much easier, the same cannot be said about QoS features. MIC and most other tools are primarily focused on the functional logic with at best limited capabilities to capture QoS concerns. Even though QoS middleware such as Quality Objects (QuO) [4, 9, 15] has made QoS adaptive design and implementation simpler, using abstractions and mechanisms provided by the middleware still 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. Without significant improvement in our ability to simplify and at least partially automate the development of adaptive QoS strategies for collections of distributed components, many applications of these complex QoS concepts and contexts will remain impractical to build or be of limited utility. Therefore, there is a compelling need for applying design-time methodologies to develop and control these runtime adaptations systematically.

In this paper, we describe the application of DQME, a modeling environment that we have created for designing QoS adaptive applications, to the design of a representative DRE application involving multiple UAVs engaged in multi-mode missions.

2 Distributed QoS Modeling Environment

Under the auspices of the DARPA MoBIES program, we applied MIC to the problems of designing, customizing, and managing the adaptive runtime characteristics of DRE applications. In particular, we utilized and augmented the Generic Modeling Environment (GME) [1] to develop the domain-specific Distributed QoS Modeling Environment (DQME) for designing runtime adaptive capabilities as provided by the QuO adaptive QoS middleware framework.

QuO is a middleware framework that supports dynamic QoS adaptation in distributed applications [9]. 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 manage dynamic adaptations. It has been used in a number of demonstrations and applications [5], ranging from wide-area distributed applications to embedded real-time systems. With QuO, distributed applications can specify (1) their QoS requirements, (2) the system elements that must be monitored and controlled to measure and provide QoS, and (3) the behavior for controlling and providing QoS and for adapting to QoS variations that occur at run-time. QuO separates the role of functional application development from the role of developing the QoS behavior of the system.

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 [2]. As a reusable framework for creating domain-specific design environments, GME supports a set of abstract modeling concepts that are generic enough to be applicable to a wide range of domains, not just limited to software systems. These concepts include containment (composition, aggregation, and hierarchy), module interconnection and interaction, aspect modeling, inheritance, and attributes. It uses metamodeling to define a domain-specific language and to model constraints and relationships. Selected concepts are customized for a target domain during the metamodeling process. A metamodel defines the family of domain models that can be created using the domain-specific modeling environment. System modelers use the resulting environment to create application models to analyze, simulate, and automatically generate the target implementation. GME provides support for tool integration and extensibility [3]. Model information and data can be communicated bi-directionally through various programming interfaces, including COM, BON (Builder Object Network), and UDM (Unified Data Model).

This work was sponsored by the Defense Advanced Research Projects Agency (DARPA) under contracts F33615-02-C-4037 and F33615-03-C-3317 with Air Force Research Laboratory, Information Directorate and Air Vehicles Directorate. Cleared for Public Release by ASC Public Affairs.

While both QoS adaptive middleware such as QuO and MIC environments such as GME are effective at what they set out to do, both also have important limitations. QoS adaptations are currently mostly hand-coded in special purpose languages and environments, limiting the complexity of the adaptation strategies to those that a QoS designer can understand in terms of localized behaviors and localized measurements while the QoS features themselves are inherently crosscutting and distributed. MIC tools such as GME can capture complex design concepts at various levels of abstraction and intelligently produce effective designs and code elements representing that design. However, they currently have limited or no high-level capabilities for representing reusable QoS-related behavioral aspects in a way that would facilitate the integrated design of QoS adaptive software. A merging of these two tools would increase the power of both, ultimately leading to tools that can be used to build integrated models of application domains and adaptation strategies.

We have developed a semantically rich, domain-specific modeling language supporting the high-level design/representation of QoS adaptation strategies and concerns based on the technical approach proposed in our earlier papers [6, 14]. Our prototype environment DQME supports hierarchical models of the essential components described in [14], with the higher levels representing the components of end-to-end QoS adaptations in support of system wide goals. The model developer can descend into these models to create models of more local adaptations, based upon local parameters, but coordinated with the other subordinate models in support of the higher level goals. The six essential model objects captured in DQME metamodel (as identified in [14]) are discussed below with focus on two components, system dynamics and adaptation strategies.

- Mission requirements – This is used to capture the high-level mission requirements of the system: functional and QoS goals that must be met by the application; relative importance of tasks; and minimum requirements on performance or fidelity. These help determine the relative importance of adaptation strategies and tradeoffs.
- Observable parameters – Before any adaptation action can be taken, the system has to know its current state with regard to the QoS of interest. These parameters determine the application's current state and they are also important to help determine and/or predict the system's next state.
- Controllable parameters and adaptation behaviors – These are 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 [8]. By selecting a set of these knobs, we can move a system from one state to another.
- System dynamics – Based on the current state of the system (Observable parameters) and the set of knobs available (Controllable parameters and adaptation behaviors), there could be several options for adaptation. The interactions between these are reflected in the

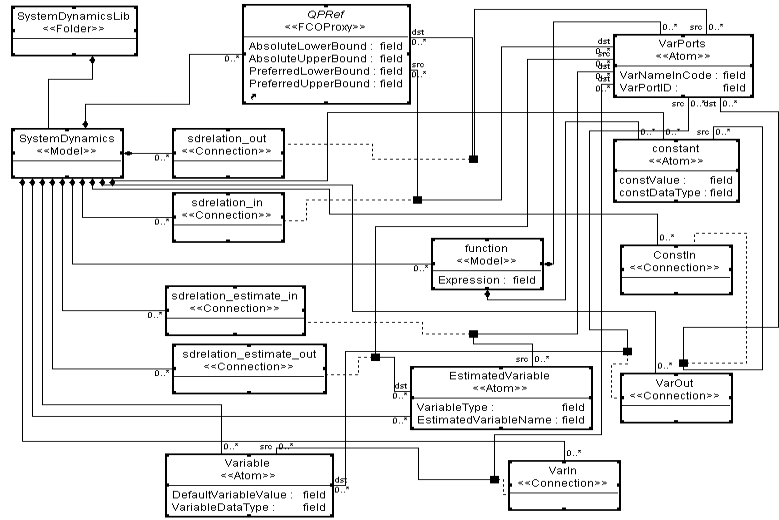


Figure 1. SystemDynamics metamodel in DQME

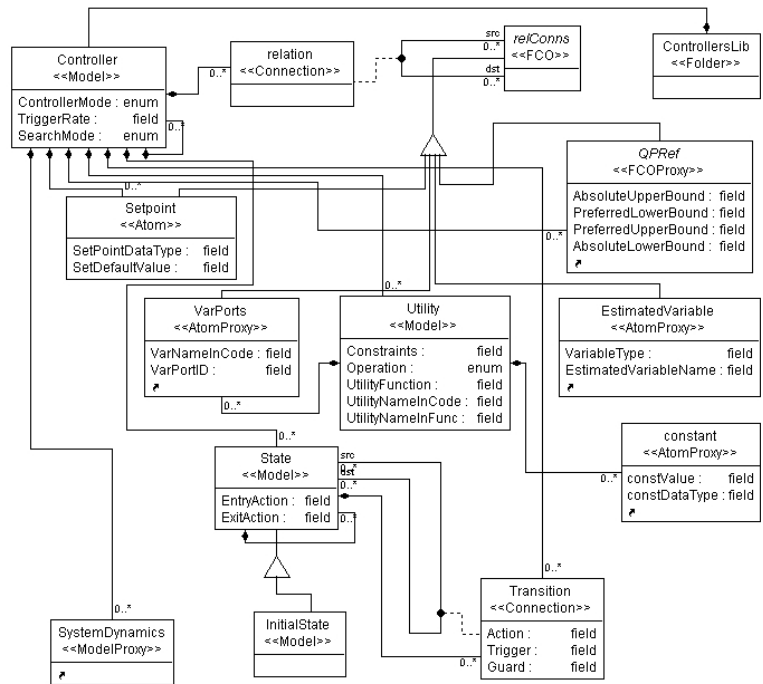


Figure 2. Controller metamodel in DQME

system dynamics and can help define the set of possible trajectories that an application can take. The *SystemDynamics* metamodel is shown in Figure 1. The *QPREf* object is a reference to *Controllable* or *Observable* object. The key of the *SystemDynamics* is the *function* object. Given some inputs, a function produces some outputs based on the logic expressed in the object. Analytical models with complicated logics can be modeled using chained functions. The outputs can then be used by the adaptation strategies to determine the next system state.

- Adaptation strategies – Here we actually choose an adaptation strategy, which specifies the adaptations employed and the tradeoffs made in response to dynamic system conditions in order to maintain an acceptable mission posture. When making a decision, the adaptation strategies take into account the mission requirements, current system state, as well as system dynamics using a utility function. The goal is to satisfy the mission requirements and maximize the utility. Multilevel adaptation strategies are modeled with a hierarchical representation at different levels of adaptation. The *Controller* metamodel is shown in Figure 2. Our metamodel currently supports the following types: state machines, supervisory control, feedback control, and search spaces. We are currently adding more types such as region spaces and classical compensators to capture a wider range of adaptation strategies. The *Utility* object provides a function to calculate the utility of the system. The goal is to maximize the utility, thus maximizing the QoS, through adaptation.

To support automated application synthesis, we developed various tools to generate configuration files, QuO code, C++ code, and Matlab code. These synthesis tools allow us to build highly complex DRE systems, with adaptive QuO code and QoS-kets inserted at appropriate places, in minutes rather than hours or days. A new set of artifacts can be regenerated to reflect the changes in the model, thus eliminating human errors in manual code modification throughout the application. We have also developed mechanisms through the UDM interface to feed runtime measurements and evaluations (e.g., statistics, optimal parameter values) back into the model, thus allowing us to do iterative model refinement.

3 Multi-UAV Surveillance and Tracking Application

As an Open Experimental Platform (OEP) and demonstration for the DARPA Program Composition for Embedded Systems (PCES) program, we have been developing a representative DRE system focused on the use of multiple reconnaissance UAVs (RUAVs) to do surveillance and target tracking in conjunction with Combat UAVs (UCAVs) and ground vehicles for time critical targeting. The scenario of the demonstration is illustrated in Figure 3 and involves a set of RUAVs performing theater-wide surveillance, sending surveillance imagery to a Combined Air Operations Center (CAOC). When a commander recognizes an item of interest (e.g., a target or threat), he commands a RUAV to concentrate its surveillance on the area of interest (AOI). When a positive identification of a threat has been made, the commander dispatches a ground or air combat unit to engage the target, with the UCAV performing battle damage assessment (BDA) afterwards.

As part of our PCES and MoBIES efforts, we are capturing the end-to-end QoS requirements necessary to support the Multi-UAV Surveillance and Tracking Application in DQME. A screen shot of the multi-UAV application modeled in DQME is shown in Figure 4. While this work is ongoing, the following sections describe our initial efforts to do this.

3.1 Mission Requirements

There are three primary *mission modes* that have to be captured, with mission requirements associated with each of them.

Mission Mode A: Surveillance. In this mode, all RUAVs are performing surveillance. The primary mission is to maximize the surveilled area with sufficient resolution in imagery for a commander to determine an item of interest. This means that imagery from each RUAV must be sent at a sufficient *rate* to ensure there are no gaps in surveillance coverage and at sufficient *size* and *resolution* for a commander to discern command level detail.

The maximum possible rate, size, and resolution are determined by the capabilities of the RUAV's camera. A representative UAV camera can provide 30 frames per second, 720x480 pixel, 24 bits per pixel imagery. This means that one UAV could provide

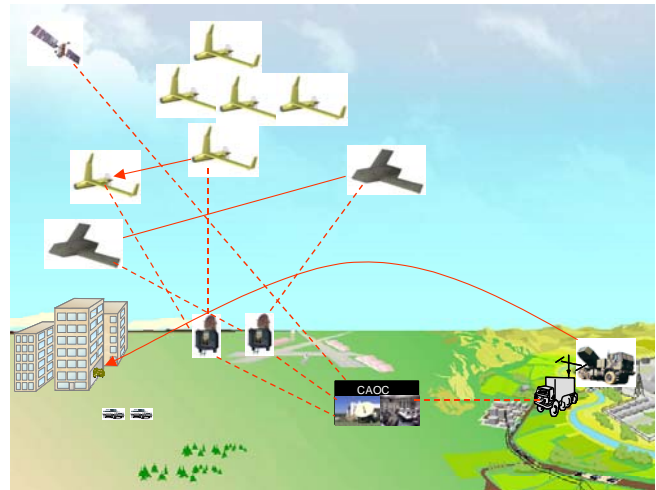


Figure 3. The OEP application consists of simulated RUAVs performing surveillance, a UCAV and ground units that can handle time critical targets and perform battle damage assessment.

- Resource attributes to measure at runtime include *bandwidth capacity*, *bandwidth used*, *CPU available*, *CPU used*, *Diff-serv Code Point values*, *CPU priorities*, and *CPU reservations (requested and achieved)*.
- Attributes of the adaptation strategies that can be measured include *compression level*, *CPU usage of compression routines*, *data size reduction due to compression*, and *number of tiles*.

3.3 Controllable Parameters and Adaptation Behaviors

We have wrapped a number of controllable QoS mechanisms and other adaptive behaviors as QuO qoskets that can be configured and assembled into the application at various places for end-to-end QoS. Each of these is represented in the model as a component and hooked to the application components, system resources, and other qosket components that it affects or with which it interacts. The following is a partial list of those we are using in the PCES OEP:

- Network priorities using Diffserv codepoints
- CPU priorities, RT CORBA priorities, and CPU Reservations (using University of Utah's CPU Broker)
- Image compression using several compression algorithms, including lossy and lossless.
- Image scaling, cropping, and tiling.
- Pacing (changing rate)

3.4 System Dynamics

The system dynamics part of the model is one of the most important, and can also be one of the most difficult to capture. It requires an analytical or experimental understanding of the ways in which the adaptation behaviors can affect the observable parameters.

Intuitively, we can capture some of the basic system dynamics as follows:

- Invoking compression will reduce the amount of bandwidth used but will increase the CPU usage (for compression and for decompression)
- Scaling and cropping will reduce the amount of bandwidth used and should increase CPU usage very little. Scaling will decrease image resolution and cropping will decrease the effective scan size.
- Lossless compression will not decrease image resolution, but lossy compression will. Lossy compression should get a higher compression ratio or take less CPU, in order to have any benefit over lossless compression.

The exact factors by which these behaviors affect the observable parameters can vary by behavior, image content and type, and due to other factors. There are two approaches that we can take to modeling the system dynamics and using them for the runtime adaptation. The first is to capture only the basic system dynamic relationship (e.g., whether an adaptation increases or decreases bandwidth usage) and develop a reactive adaptation strategy that chooses (or guesses) an adaptation to invoke and monitors the change to the system, reactively changing or adjusting the adaptation to improve the performance.

Since our application requires real-time behavior and our requirements are mission critical ones, we employ a second approach to developing our system dynamics. We plan to conduct a series of experiments using imagery gathered from the RUAV and UCAV cameras to determine the affect of the various compression and image manipulation behaviors we have. We will use the results of these experiments to choose the factors of the system dynamic formulas, add in a little "wiggle room," and supplement this at runtime with an adaptation engine that will make minor adjustments to handle slight range violations.

While these system dynamics are critical for our adaptation strategies, there are other important ones for this application that we can more fully represent, such as the following:

$$\begin{aligned} \text{total_BW_used (bps)} &= \text{size_of_RUAV_imagery (bps)} + \\ &\text{size_of_UCAV_imagery (bps)} + \text{size_of_control_traffic (bps)} \end{aligned} \quad (1)$$

$$\text{end2end_latency} = \sum \text{processing}(n_i) + \sum \text{transmit}(e_i) \quad (2)$$

where $\text{processing}(n_i)$ is the time spent processing for a stream on node n_i and $\text{transmit}(e_i)$ is the time to send an image across link e_i for a given stream.

3.5 Adaptation Strategies

In all modes, approximately 10% of the network bandwidth and CPU processing will be set aside for control traffic (i.e., the pushing of policy, mission mode changes, position updates, and other command and control traffic). The resource manager at the CAOC will allocate a specific amount of bandwidth and CPU to each RUAV and UCAV based on the system dynamics discussed in section 3.4. In Mode A, all RUAVs will evenly divide all the remaining available bandwidth and CPU. In mode B and C, the privileged RUAV and the UCAV will be allocated a higher amount of bandwidth and CPU and all other non-privileged RUAVs

will evenly divide the remaining resources as in Mode A. Each non-privileged RUAV will use a local adaptation strategy to select a frame rate, image size, and compression function such that the following constraint is satisfied:

$$\text{used_BW}(U_i) \leq \text{allocated_BW}(U_i) \wedge \text{used_CPU}(U_i) \leq \text{allocated_CPU}(U_i) \quad (3)$$

We can model the RUAV adaptation strategy in two ways. First, we could tradeoff the rate, size, and compression level equally in steps (in round robin fashion in the order specified by the mission requirements) until the constraint is met. Alternatively, the strategy can reduce the rate until the constraint is satisfied or the minimum is reached, followed by the image size until the constraint is met or the minimum is reached, and then the compression.

The privileged RUAV and UCAV adaptation strategies will set their Diffserv codepoints to “privileged” class and request sufficient CPU reservations. The privileged RUAV adaptation strategy will then reduce the imagery rate, followed by invoking lossless compression, followed by image cropping, each until the minimum or until the constraint in (3) is met. The UCAV adaptation strategy will select a fixed rate for the BDA imagery (probably one frame every four seconds) and invoke lossless compression, followed by tiling of the imagery until the constraint in (3) is satisfied.

4 Conclusions

We have developed the DQME modeling environment, based on GME and QuO, for designing the QoS adaptation necessary for DRE applications. As part of a capstone demonstration for the DARPA PCES program, we are evaluating DQME’s suitability to model representative DRE systems by modeling the end-to-end QoS adaptation for a multi-UAV surveillance and target tracking application. This work is ongoing and is not without significant challenges, but shows promise for facilitating the design and top-down construction of DRE applications, even while application to the PCES OEP shows promise for helping us to mature and evaluate the usefulness of DQME.

References

- [1] Ledeczki A. GME 4 Users Manual (v4.0), 2004. Web Site: <http://www.isis.vanderbilt.edu/Projects/gme/GMEUMan.pdf>.
- [2] Ledeczki, A., Maroti, M., Bakay, A., Karsai, G., Garrett, J., Thomason, C., Nordstrom, G., Sprinkle, J., and Volgyesi, P. The Generic Modeling Environment. WISP'2001, Budapest, Hungary, May 24-25 2001.
- [3] Ledeczki, A., Bakay, A., Maroti, M., Volgysei, P., Nordstrom, G., Sprinkle, J., and Karsai, G. Composing Domain-Specific Design Environments. IEEE Computer, 34,11(Nov. 2001), 44 – 51.
- [4] Loyall, J., Schantz, R., Zinky, J., and Bakken, D. Specifying and Measuring Quality of Service in Distributed Object Systems. In Proceedings of The 1st IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC 98), April 20-22, 1998, 43-52.
- [5] Loyall, J., Schantz, R., Zinky, J., Pal, P., Shapiro, R., Rodrigues, C., Atighetchi, M., Karr, D., Gossett, J.M., and Gill, C.D. Comparing and Contrasting Adaptive Middleware Support in Wide-Area and Embedded Distributed Object Applications. In Proceedings of the 21st IEEE International Conference on Distributed Computing Systems (ICDCS-21), April 16-19, 2001, 625-634.
- [6] Loyall, J., Shapiro, R., Neema, S., Abdelwahed, S., Schantz, R., and Mahadevan, N. Model-Based Design of Runtime Adaptation Strategies. RTAS 2003 Workshop on Model-Driven Embedded Systems (RTAS MoDES), Tuesday May 27, 2003 at RTAS 2003 Washington, DC, May 27-30, 2003.
- [7] Object Management Group. Common Object Request Broker Architecture (CORBA): Core Specification. . OMG Document formal/04-03-01, March 2004.
- [8] Schantz, R., Loyall, J., Atighetchi, M., and Pal, P. Packaging Quality of Service Control Behaviors for Reuse. In Proceedings of the 5th IEEE International Symposium on Object-Oriented distributed Computing (ISORC 02), April 29-May1 2002.
- [9] Schantz, R. E., Loyall, J., Rodrigues, C., Schmidt, D.C., Krishnamurthy, Y., and Pyarali, I. Flexible and Adaptive QoS Control for Distributed Real-time and Embedded Middleware. The ACM/FIP/USENIX International Middleware Conference, June 2003, Rio de Janeiro, Brazil.
- [10] Sztipanovits, J. and Karsai, G., Model-integrated computing. Computer 30, 4(April, 1998), 110-111.
- [11] Thai, T. and Lam, H. .Net Framework Essentials. O'Reilly, Cambridge, MA. 2001.
- [12] Wang, N., Schmidt, D.C., Gokhale, A., Gill, D., Natarajan, B., Rodrigues, C., Loyall, J.P., and Schantz, R.E. Total Quality of Service Provisioning in Middleware and Applications. The Journal of Microprocessors and Microsystems, Elsevier, 26, 9-10, (Jan 2003).
- [13] Wollrath, R.R. and Waldo, J. A Distributed Object Model for the Java system. USENIX Computing Systems 9,4(Fall 1996).
- [14] Ye, J., Loyall, J., Shapiro, R., Neema, S., Mahadevan, N., Abdelwahed, S., Koets, M., and Varner, D. A Model-Based Approach to Designing QoS Adaptive Applications. 2004. Submitted for publication.
- [15] Zinky, J., Bakken, D., and Schantz, R. Architectural Support for Quality of Service for CORBA Objects. Theory and Practice of Object Systems 3(1). 1997.