Concern Separation for Adaptive QoS Modeling in Distributed Real-Time Embedded Systems

Jeff Gray[1], Sandeep Neema[2], Jing Zhang[1], Yuehua Lin[1],
Ted Bapty[2], Aniruddha Gokhale[2], Douglas C. Schmidt[2]

[1] Dept. of Computer and Information Sciences, University of Alabama at Birmingham
Birmingham AL 35294-1170
{gray, zhangj, liny} @ cis.uab.edu
http://www.gray-area.org

[2] Institute for Software Integrated Systems, Vanderbilt University
Nashville TN 37211
{sandeep, bapty, gokhale, schmidt} @ isis.vuse.vanderbilt.edu
http://www.isis.vanderbilt.edu

Abstract. The development of Distributed Real-Time and Embedded (DRE) systems is often a challenging task due to conflicting Quality of Service (QoS) constraints that must be explored as trade-offs among a series of alternative design decisions. The ability to model a set of possible design alternatives, and to analyze and simulate the execution of the representative model, offers great assistance toward arriving at the correct set of QoS parameters needed to satisfy the requirements for a specific DRE system. This paper presents a model-driven approach for generating QoS adaptation in DRE systems. The approach involves the creation of high-level graphical models representing the QoS adaptation policies. The models are constructed using a domain-specific modeling language - the Adaptive Quality Modeling Language – which presents a view-oriented separation of common concerns. The paper motivates the need for aspect weavers at the modeling level, and provides a case study that is based on bandwidth adaptation in video streaming of an Unmanned Aerial Vehicle (UAV).

1. Introduction

Rapid advances in middleware technologies have given rise to a new generation of highly complex, object-oriented Distributed Real-Time andEmbedded (DRE) systems based on platforms such as RT-CORBA, COM+, RMI, among others. As observed in [26], middleware solutions promote software reuse resulting in higher development productivity. However, within the context of DRE systems, specifying and satisfying quality-of-service (QoS) requirements commonly use ad-hoc, problem-specific code optimizations. Such modifications defy the very principles of reuse and portability.

There have been a few attempts to improve this situation by introducing a programmable QoS adaptation layer on top of the middleware infrastructure. The key idea behind the adaptation layer is to offer *separation of concerns* with respect to QoS requirements. Such separation provides improved modularization for separating QoS requirements from the functional parts of the software. As an example, the Quality Objects (QuO) project, developed by BBN, is one such adaptation layer [27]. QuO provides an extension to the Object Management Group's (OMG) Interface Definition Language (IDL). This extension, known as the Contract Definition Language (CDL), supports the specification of QoS requirements and adaptation policies. Contracts written in CDL are compiled to create stubs that monitor and adapt the QoS parameters when the system is operational. These stubs are integrated into the QuO kernel.

1.1 Problems Endemic to QoS Implementation

From a software engineering standpoint the CDL approach works well; however, there are a few drawbacks to using CDL alone for systems development:

1. The control-centric nature of the QoS adaptation extends beyond software concepts. Issues such as stability and convergence are paramount. In a DRE system, QoS is specified in software parameters, which have a significant impact on the dynamics of the overall physical system. Owing to the complex and non-linear dynamics, it is very difficult to tune the QoS parameters in an ad-hoc manner without compromising the stability of the underlying physical system. The QoS adaptation software is, in effect, equivalent to a controller for a discrete, non-linear system. Therefore, sophisticated tools are needed to design, simulate, and analyze the QoS adaptation software from a control system perspective.
2. For scalability reasons, the CDL offers a much lower level of abstraction (textual code-based). Even for a moderately large system, the CDL specifications can grow quite large. Implementing even a small change to the adaptation policy requires making several changes manually, while ensuring that all these changes are consistent in the CDL specification. In addition, small changes can have far-reaching effects in the dynamic behavior due to the nature of emergent crosscutting properties.

1.2 A Solution: Aspects for Supporting Model-Driven Design

In this paper, the above issues are addressed by applying the principles of Aspect-Oriented Software Development (AOSD) [18, 20, 31] to Model-Driven Design [2, 10]. From a modeling perspective, expressive power in software specification is often gained from using notations and abstractions that are aligned to a specific problem domain. This can be further enhanced when graphical representations are provided to model the domain abstractions. In domain-specific modeling, a design engineer describes a system by constructing a visual model using the terminology and concepts from a specific domain [17]. Analysis can then be performed on the model, or the model can be synthesized into an implementation. Model-Integrated Computing (MIC) has been refined over the past decade at Vanderbilt University to assist in the creation and synthesis of complex computer-based systems [29]. A key application area for MIC is in those systems that have a tight integration between the computational structure of a system and its physical configuration (e.g., embedded systems). In such systems, MIC has been shown to be a powerful tool for providing adaptability in evolving environments. The Generic Modeling Environment (GME) [19] is a domain-specific modeling tool that realizes the principles of MIC. The GME provides meta-modeling capabilities that can be configured and adapted from meta-level specifications (representing the modeling paradigm) that describe the domain. As shown in Figure 1, Model-Driven Development of a DRE system using MIC involves the following three activities:

- **Design**: The approach described in this paper uses a graphical modeling environment that allows a designer to model the DRE system and the QoS adaptation policy using standardized notations, such as Statecharts [14] and Dataflow (this is the Design phase of Figure 1). The modeling language is partitioned among various perspectives that allow the modeler to focus on specific related views of the design. Aspect-oriented weavers at the modeling level assist the modeler in rapidly changing crosscutting properties of a model that are traditionally difficult to change due to the scattering of their specification (e.g., a policy for adapting bandwidth usage that spans across multiple states in a finite state machine (FSM)).
- **Synthesis**: As represented by the "Synthesis" part of Figure 1, a generator tool synthesizes artifacts for Matlab Simulink/Stateflow® (a popular commercial tool) providing the ability to simulate and analyze the QoS adaptation policy. This gives significant assurance that the system will perform as desired without having to deploy the actual system in the field.
- **Execution**: A second generator tool creates CDL specifications from the QoS adaptation models. The generated CDL is then compiled into executable artifacts (the "Execution" phase of Figure 1). The right-side of Figure 1 illustrates the screen-shots taken from the execution of a target recognition system that was generated by a model representation. By modeling the DRE system at a higher-level of abstraction, a modeler can evolve the design while not getting involved in the accidental complexities of specific implementation techniques. A previous study showed that small changes to an abstraction represented at the modeling level resulted in large changes across the corresponding CDL representation [22].

This paper focuses attention on the Design phase of Figure 1. The approach described in this paper has benefits that are similar to those of the OMG's Model-Driven Architecture (MDA) [2, 10], Model-Driven Middleware (MDM) [11] and Adaptive-Reflective Middleware (ARM) [26]. The unique contribution of the paper is found in the utilization of an aspect-oriented weaver that performs model transformations across higher-level abstractions, such as FSMs, to separate policy decisions that were previously scattered and tangled.

The rest of this paper is organized as follows. Section 2 presents the modeling concepts (i.e., interaction among related metamodels) and environment to support modeling viewpoints for QoS adaptation in DRE systems. Section 3 introduces a video streaming application that is represented using the modeling language of Section 2. The video streaming application is used as a case study of aspect-oriented modeling in Section 4. The paper concludes with summary remarks in Section 5.

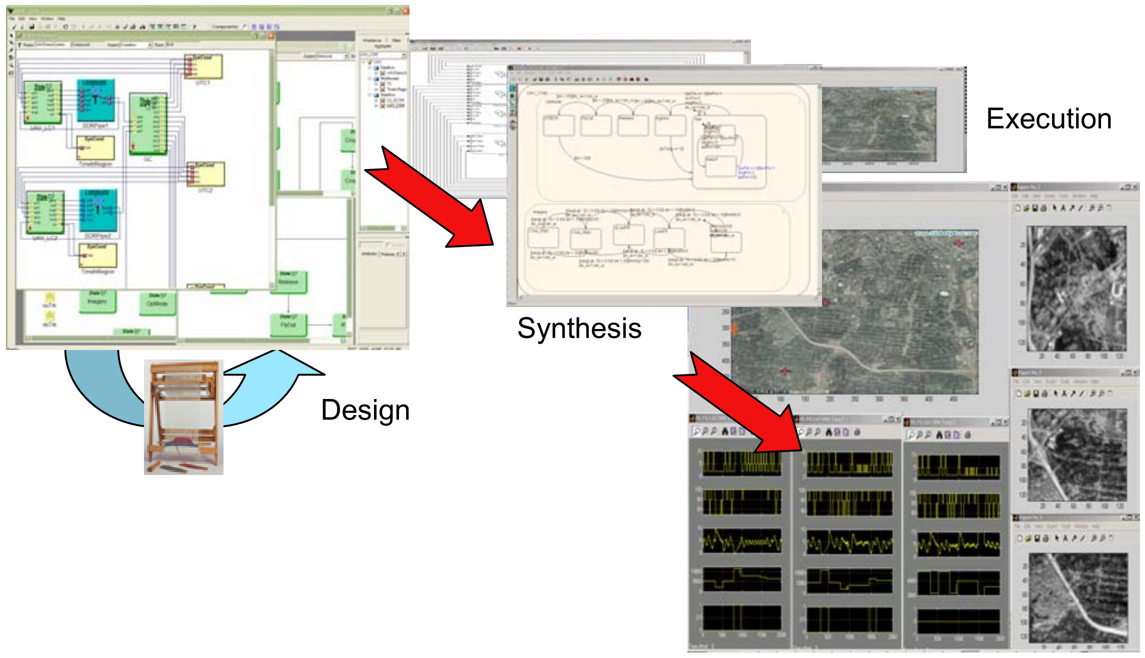


Figure 1: Model-Driven Design of Adaptive QoS for DRE Systems

2. Viewpoint Modeling for Specifying QoS Adaptation in DRE Systems

The MIC infrastructure provides a unified software architecture and framework for creating a customized domain-specific modeling environment [19, 29]. The core components of the MIC infrastructure are: a customizable *Generic Model Editor* for creation of multiple-view, domain-specific models; *Model Databases* for storage of the created models; and, a *Model Interpretation* technology that assists in the creation of domain-specific, application-specific model interpreters for transformation of models into executable/analyzable artifacts. The Generic Modeling Environment (GME) is a meta-configurable modeling tool that implements the goals of MIC [19]. The GME includes tools and functionality to support the creation and storage of system models, in addition to generation of executable/analyzable artifacts from these models. For almost a decade, the GME has been freely available (please see http://www.isis.vanderbilt.edu/Projects/gme/) and used by multiple research groups on several dozen industrial and federally-sponsored research projects.

In the MIC technology, the modeling concepts to be instantiated in the MIPS environment are specified in a *meta-modeling* language [17]. A *metamodel* of the modeling paradigm is constructed that specifies the syntax, static semantics, and the presentation semantics of the domain-specific modeling paradigm. The metamodel uses a Unified Modeling Language (UML) class diagram to capture information about the objects that are needed to represent the system information and the inter-relationships between different objects. The meta-modeling language also provides for the specification of visual presentation of the objects in the graphical model editor [19].

The Adaptive Quality Modeling Language (AQML), specified as a GME modeling paradigm, assists in modeling key aspects of a DRE system. Each area of concern in the model is partitioned into a specific view that slices through a particular perspective of the overall model. The separation of concerns provided by a GME view is similar in intent to previous research on viewpoints [23] in requirements engineering. In AQML, the three views defined in the metamodel are:

1. **QoS Adaptation Modeling**: In this first category, the adaptation of QoS properties of the DRE system is modeled. The designer can specify the different state configurations of the QoS properties, the legal transitions between the different state configurations, the conditions that enable these transitions (and the actions that must be performed to enact the change in state configuration), the data variables that receive and update QoS information, and the events that trigger the transitions. These properties are modeled using an extended finite-state machine (FSM) modeling formalism [14].
2. **Computation Modeling**: In this category, the computational aspect of a DRE system is modeled. A dataflow model is created in order to specify the various computational components and their interaction. An extended dataflow modeling formalism is employed.
3. **Middleware Modeling**: In this category, the middleware services, the system monitors, and the tunable "knobs" (i.e., the parameters being provided by the middleware) are modeled.

The metamodel of each of these modeling categories and their interaction in the AQML is described throughout the remainder of this section. Each metamodel represents a view of the overall system, and each metamodel is linked together through interactions (Section 2.4)

2.1 QoS Adaptation Modeling

Stateflow models capture the adaptive QoS behavior of the system. A discrete finite State machine representation, extended with hierarchy and concurrency, is selected for modeling the QoS adaptive behavior of the system. This representation has been selected due to its scalability, universal acceptability, and ease-of-use in modeling. Figure 2 illustrates the QoS adaptation view of the AQML.

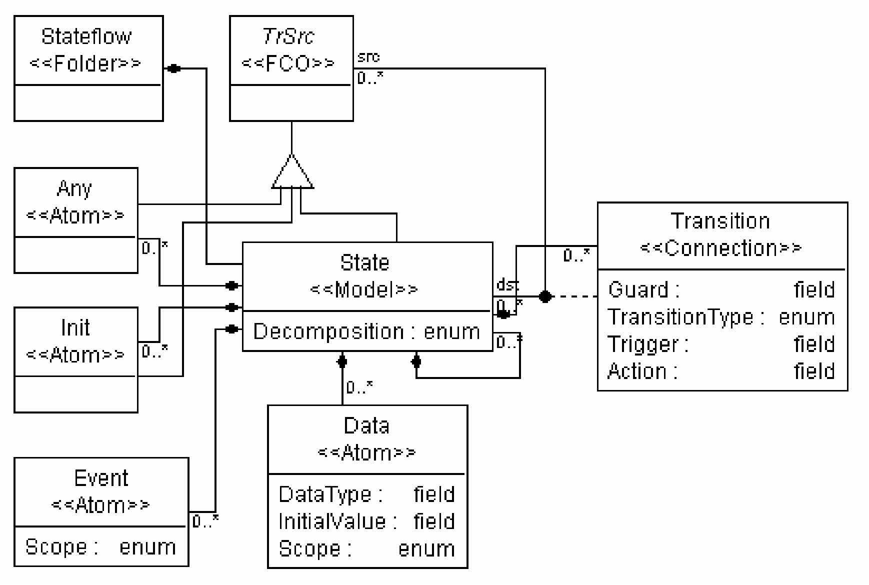


Figure 2: Metamodel of QoS Adaptation Modeling

The main concept in a Finite State Machine (FSM) representation is a *state*. States define a discretized configuration of QoS properties. Hierarchy is enabled in the representation by allowing States to contain other States. Attributes define the decomposition of the State. The State may be an *AND* state (when the state machine contained within the State is a concurrent state machine), or, the State can be an *OR* state (when the state machine contained within the State is a sequential state machine). If the State does not contain child States, then it is specified as a *LEAF* state. States are stereotyped as *models* in the MIC framework.

Transition objects are used to model a transition from one state to another. The attributes of the transition object define the trigger, the guard condition, and the actions. The trigger and guard are Boolean expressions. When these Boolean expressions are satisfied, the transition is enabled and a state change (accompanied with the execution of the actions) takes place. Transitions are stereotyped as a *connection* in the GME tool. To denote a transition between two States, a connection has to be made from the source state to the destination state.

In addition to states and transitions, the FSM representation includes data and events. These can be directly sampled external signals, complex computational results, or outputs from the state machine. In the modeling paradigm, *Event* objects capture the Boolean event variables, and the *Data* objects capture arbitrary data variables. Both the Events and Data have a Scope attribute that indicates whether an event (or data) is either local to the state machine or is an input/output of the state machine.

2.2 Computation Modeling

This view is used to describe the computational architecture. A dataflow representation, with extensions for hierarchy, has been selected for modeling computations. This representation describes computations in terms of computational components and their data interactions. To manage system complexity, the concept of hierarchy is used to structure the computation definition. Figure 3 illustrates the computation view of the AQML. The different objects and their inter-relationships are described below.

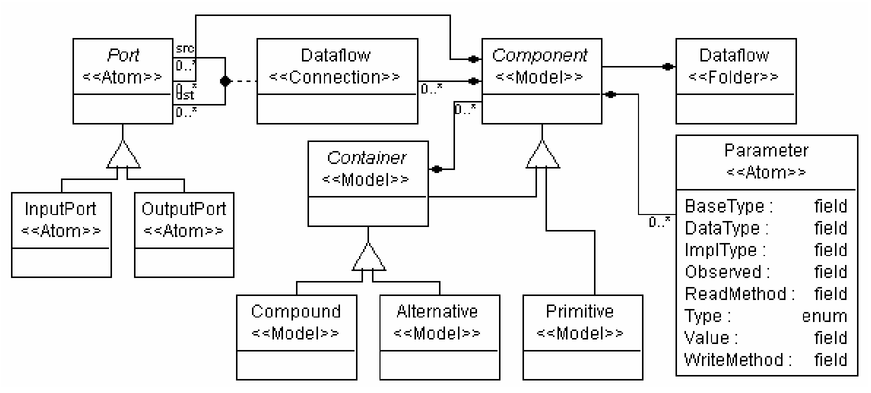


Figure 3: Metamodel of Computation Architecture Modeling

The computational structure is modeled with the following classes of objects: *Compounds*, *Alternatives*, and *Primitives*. These objects represent a computational component in a dataflow representation. *Ports* are used to define the interface of these components through which the components exchange information. Ports are specialized into *InputPorts* and *OutputPorts*.

A Primitive is a basic modeling element that represents an elementary component. A Primitive maps directly to a processing component that will be implemented as a software object or a function. A Compound is a composite object that may contain Primitives or other Compounds. These objects can be connected within the compound to define the dataflow structure. Compounds provide the hierarchy in the structural description that is necessary for managing the complexity of large designs. An Alternative captures "design choices" – functionally equivalent designs for a rigorously defined interface, providing the ability to model design spaces instead of a single design.

An important concept relevant to QoS adaptive DRE systems is the notion of parameters. Parameters are the tunable "knobs" that are used by the adaptation mechanism to tailor the behavior of the components such that desired QoS properties are maintained. Parameters can be contained in Compounds and Primitives. The Type attribute defines whether a Parameter is read-only, write-only, or read-write. The DataType attribute defines the data type of the parameter.

2.3 Middleware Modeling

In this view, the components of the middleware are modeled. These components include the services and the system conditions provided by the middleware. Examples of services include an Audio-Video Streaming service, Bandwidth reservation service, Timing service, and Event service, among others. System conditions are components that provide quantitative diagnostic information about the middleware. Examples of these include observed throughput, bandwidth, latencies, and frame-rates. Figure 4 illustrates the middleware modeling view of the AQML.

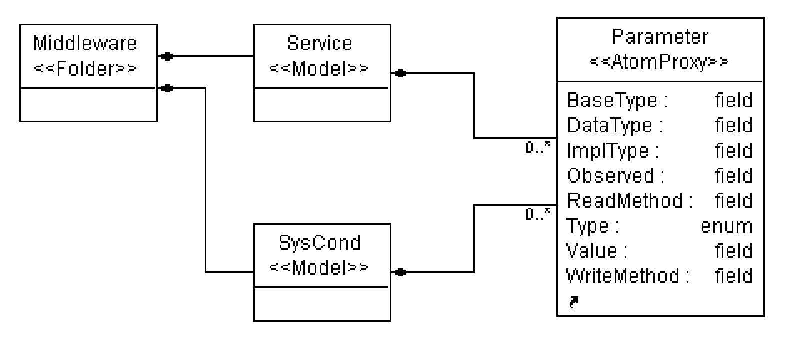


Figure 4: Metamodel of Middleware Modeling

The *Service* object represents the services provided by the middleware. Services can contain parameters that are the tunable knobs provided by the service. In addition to being tunable “knobs,” parameters play a second role as instrumentation, or probes, by providing some quantitative information about the service. The *SysCond* object represents the system condition objects present in the middleware layer. SysConds can also contain parameters.

Observe that we do not facilitate a detailed modeling of the middleware components or the dataflow components. This is because the focus of AQML is on the QoS adaptation. We model only those elements of the dataflow and middleware that facilitate the QoS adaptation (namely, the tunable and observable Parameters).

2.4 Interaction of QoS Adaptation with Middleware and Computation Modeling

In this category, the interaction of the previous three modeling categories (illustrated in Figure 5) is specified. As described earlier, the Data/Event objects within the Stateflow model form the interface of the state machine. Within the Computation and the Middleware views, Parameters form the control interfaces. The interaction of the QoS adaptation (captured in Stateflow models), and the middleware and application (modeled in the Middleware/Computation models), is through these interfaces. The interaction is modeled with the *Control* connection class, which connects the Data object of a State model to a Parameter object of a Middleware/Computation model.

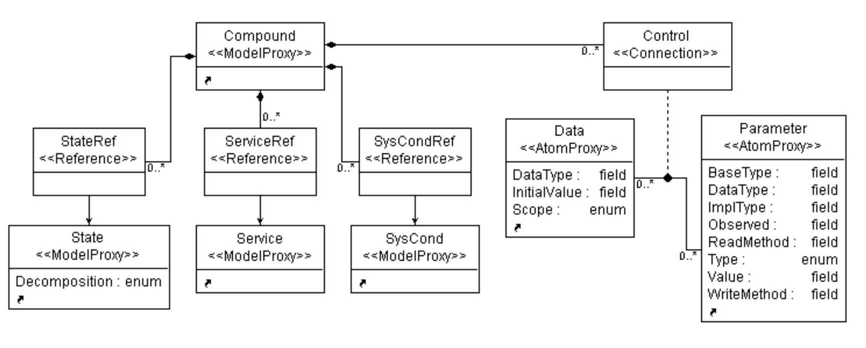


Figure 5: Interaction Metamodel of QoS Adaptation with Middleware/Computation Modeling

In the MIC framework, connections between objects that are not contained within the same model are not supported. Therefore, we create references, which are equivalent to a "reference" or a "pointer" in a programming language [17]. These are contained in the same context (model) such that a connection can be drawn between them. The *StateRef*, *ServiceRef*, and the *SysCondRef* objects are the reference objects that are contained in a Compound (Computation) Model. All of the views mentioned in this section partition the concerns of a larger model space into cohesive units that make the modeling activity more manageable.

The three metamodels defined in this section correspond specify specific views of a DRE system. At the instance model level, this provides a modeler with multiple perspectives for separating the different concerns of the model such that details not pertinent to the modeling task at hand are abstracted into other views. This offers a valuable modeling construct, but it will be shown in Section 4 that viewpoints alone are not sufficient for capturing certain types of crosscutting concerns. Before describing that issue, however, the next section introduces the problem domain to be used in the case study of the paper, which is built as an instance of the AQML metamodel.

3. Specifying QoS Policies for Adaptation of Video Bandwidth

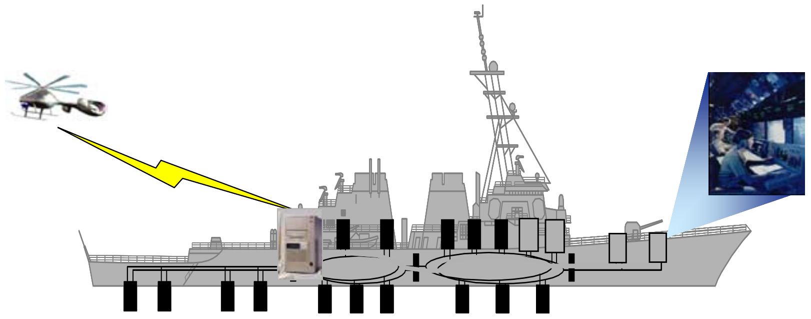


Figure 6: UAV Video Streaming Application
(reprinted from [16], with permission from BBN)

The ability to adapt is an essential trait for DRE solutions, where the presence of Quality of Service (QoS) requirements demands that a system be able to adjust in a timely manner to changes imposed from the external environment. To provide adaptability within distributed real-time systems, there are three things that must be present: 1) the ability to express QoS requirements in some form; 2) a mechanism to monitor important conditions that are associated with the environment; and 3) a causal relation between the monitoring of the environment and the specification of the QoS requirements in such a way that there is a noticeable change in the behavior of the system as it adapts [16].

The case study used in this paper demonstrates an application of QoS modeling as applied to a conceptual scenario involving a number of Unmanned Aerial Vehicles (UAV-s) conducting surveillance in a military theater. The UAV prototype has been developed by researchers from BBN as an integration test-bed and experimental platform for the DARPA Program Composition for Embedded Systems (PCES) project. The UAV streams video back to a central distributor that forwards the video on to several different displays [21]. The essence of this project is pictorially shown in Figure 6. The feedback cycle for utilizing the UAV as a surveillance device is: 1) video from the UAV is sent to the distributor that is located on a ground station or sea vessel, 2) the distributor broadcasts the video to numerous video display hosts on board the ship, 3) the video is received by each host and displayed to various operators, and 4) each operator at a display observes the video and sends commands, when deemed necessary, to control the UAV [16].

The concept of operation for the scenario can be summarized briefly as follows. Initially, several UAVs are conducting surveillance in a battlefield. UAV imagery must be transmitted in real-time to ground stations, ensuring an acceptable image quality. The latency of the transmission of an image and its reception at the ground station must be low enough to provide a continuous update of the battlefield as the UAV-s loiter over the theatre. The available bandwidth must be shared uniformly over all of the collaborating UAV-s, and each UAV must adapt its transmission rate to the available bandwidth such that the timeliness requirement of the delivered imagery is met. The scenario advances to the next stage when one or more of the UAV-s observe a target in their field-of-view. In this next stage of the scenario, QoS requirements evolve as the UAV-s observing the target acquire tactical importance. It is required that the UAV-s observing a potential target receive preferential treatment in bandwidth distribution such that the tactically important information is not delayed. All other UAV-s that are not observing a target must reduce their bandwidth usage. In this case study, attention is restricted to just these two stages of the scenario in order to keep the description apprehensible (although there are additional stages that exercise a broader spectrum of QoS adaptation).

In the presence of changing conditions in the environment, the fidelity of the video stream must be maintained according to specified QoS parameters. The video must not be stale, or be affected by jittering, to the point that the operator cannot make an informed decision. Within the BBN implementation, a *contract* assists the system developer in specifying QoS requirements that are expected by a client and provided by a supplier. Each contract describes operating regions and actions that are to be taken when QoS measurements change. A domain-specific language (DSL) was developed to assist in the specification of contracts; the name of this DSL is the Contract Description Language (CDL) [16, 27]. A code generator translates the CDL into code that is integrated within the run-time kernel of the application. The textual intention of a CDL specification is very similar to the semantics of a hierarchical state machine. The overall approach adopted by BBN for implementation of QoS adaptation is strongly aspect-oriented [7].

There are several things that make the UAV project a complex and challenging problem (i.e., the real-time requirements, resource constraints, and the distributed nature). There are a few interesting observations to make: 1) the link between the UAV to the ship is a wireless link imposing some strict bandwidth restrictions; 2) there is a need for prioritization between different video streams coming from different UAVs owing to the region of interest, nature of threat, etc; 3) latency is a higher concern than throughput because it is important to get the latest changes in the threat scenario at the earliest possible time; 4) there may be a wide-variety of computational resources (processors, networks, switches) involved in the entire application; and 5) the scenario is highly dynamic (i.e., UAVs frequently enter and leave the battle field). Given these complex requirements, a QoS-enabled middleware solution has been proposed for this application [16, 27]. The AQML modeling language was developed in response to the need for a modeling language to represent QoS specification for the UAV application. The next subsection introduces some instances of the AQML metamodel that was defined in Section 2.

3.1 AQML Modeling of the BBN UAV-s

Figure 7 shows the top-level computational components for one UAV and its interactions. The view shown is a dataflow diagram with interactions between the UAV adaptation controller, the middleware system condition variables, and the computational components responsible for transmission of video content. The box labeled "SDRPipe1" represents the top-level of the hierarchical composition of the Sender-Distributor-Receiver components that are responsible for production, distribution, and display of the video stream. The "UAV_LC1" component represents the top-level of the UAV QoS adaptation controller (detailed further below). The "C2_GC" component represents the top-level of the Ground Station's QoS adaptation controller. This is responsible for coordinating between UAV-s for distribution of bandwidth according to the tactical importance of each UAV. This particular view reveals only a single UAV. The "OTNR1," "ITNR1," "GTC," and "TC1" are system condition variables, which are middleware objects responsible for communicating QoS information across different objects in a distributed system. The OTNR /ITNR represents time in region, which keeps track of the time the UAV has been in a particular mode of operation. The TC1, and GTC keeps track of the observation of target by the UAV-s. The TC1 is a variable set by the UAV1, when it sees a target. The GTC is a logical-OR of all the TC variables, indicating if one or more of the UAV-s have observed a target. All the lines shown in view represent flow of information/data among these components. The UAV_LC1 controls the video stream parameters such as frame-rate, frame-size, and image-quality, which is indicated by the connections between the UAV_LC1's and the SDRPipe1's appropriately named ports.

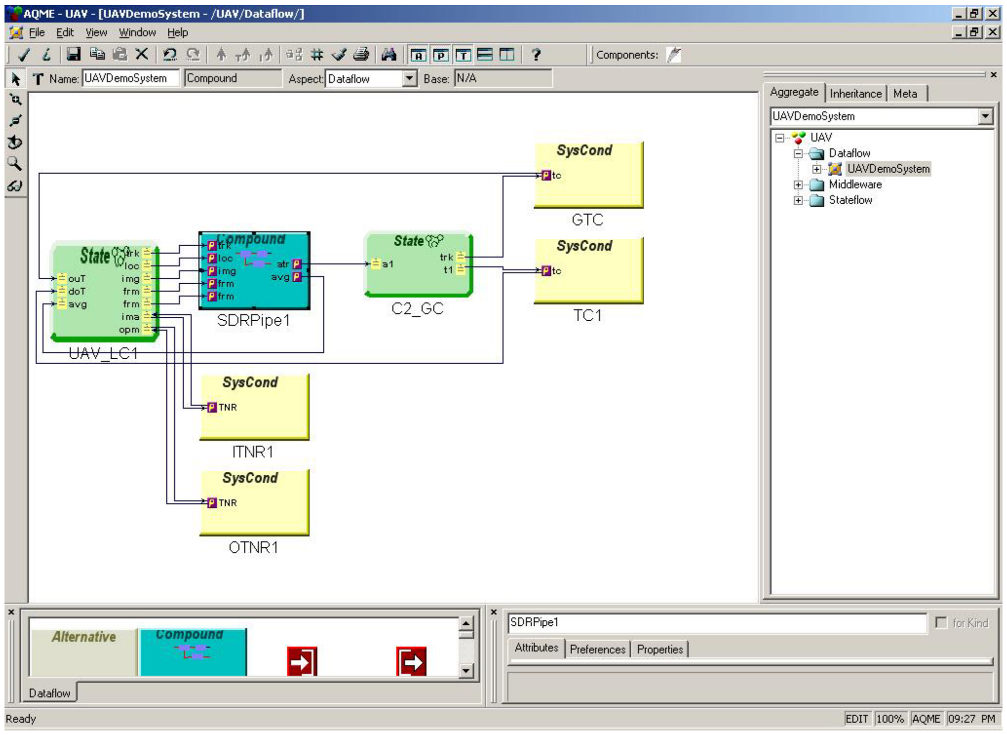
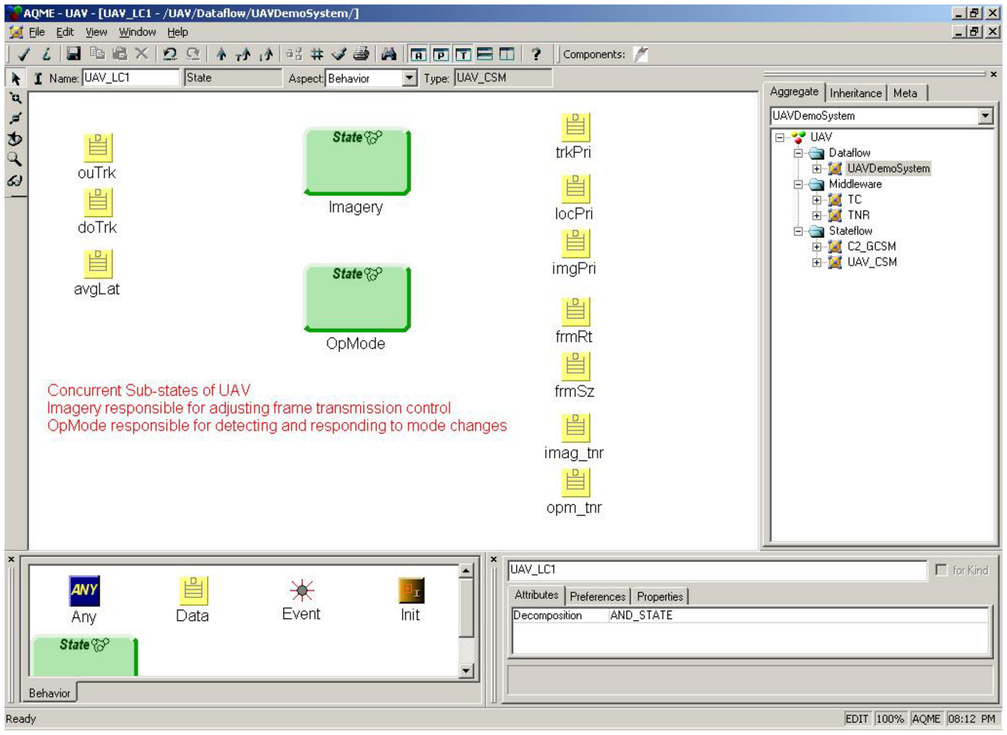


Figure 7: Top-Level Computational Components for One UAV

Figure 8 and Figure 9 show a model of the QoS-adaptive behavior of the UAV-s as modeled in AQML. Figure 8 shows the top-level behavior consisting of two concurrent states: Imagery and Op-Mode. The OpMode state represents the operational modes of the UAV related to the observation of a target. The Imagery state manages the adaptation of imagery transmission based on the available bandwidth and the operational mode. Figure 9 shows the sub-states of the Imagery state.



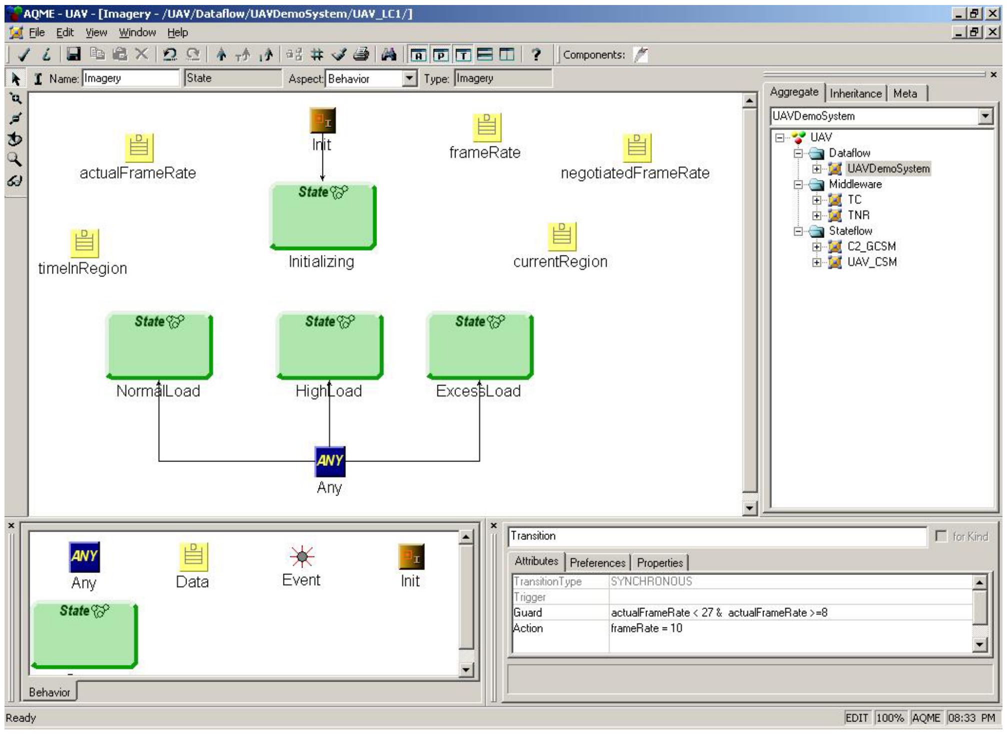
Name: UAV_LC1 | State | Aspect: Behavior | Type: UAV_CSM

Concurrent Sub-states of UAV
Imagery responsible for adjusting frame transmission control
OpMode responsible for detecting and responding to mode changes

UAV_LC1

Decomposition	AND_STATE

Figure 8: Concurrent States within one UAV



Name: Imagery | State | Aspect: Behavior | Type: Imagery

Transition

TransitionType	SYNCHRONOUS
Trigger	
Guard	actualFrameRate < 27 & actualFrameRate >=8
Action	frameRate = 10

Figure 9: Model of QoS Adaptation within Imagery State

In this application, the goal of QoS adaptation is to minimize the latency on the video transmission. When the communication resources are nominally loaded, it may be possible to transmit the video stream at the full frame rate with a minimal basic network delay. However, when the communication and computational resources are loaded, the delays in transmission expand for the same frame rate resulting in increased latencies. The adaptation scheme attempts to compensate for the increased load by reducing the rate of transmission, thus improving the latency again. There are several ways of reducing the transmission rate: a) reduce the frame rate by dropping frames, b) reduce the image quality per frame, or c) reduce the frame size. Depending on the actual scenario, one or more of these situations may apply. In the example of this section, dropping the frame rate is the only option considered.

Figure 9 shows a QoS adaptation model of the UAV scenario in the AQML. The three states NormalLoad, ExcessLoad, and HighLoad capture three different QoS configurations of the system. A few data variables (actualFrameRate, frameRate, timeInRegion) can also be seen in this figure. These data variables provide the state-machine with sensory information about the network. At the same time, other data variables may enact the adaptation actions that are being performed in the transitions. Notice that the attribute window at the bottom-right corner of Figure 9 shows the trigger, guard, and action attributes of a transition. An example guard expression is visible in the attribute window of the figure (i.e., "actualFrameRate < 27 and actualFrameRate >= 8"). When this expression evaluates to true, the transition is enabled and the modeled system enters the HighLoad state. An example action expression can be seen in this figure (i.e., "frameRate = 10"). This sets the frameRate data variable to a value of 10.

The next section provides a case study of aspect model weaving as applied to the UAV scenarios.

4. Case Study: Aspect Model Weaving for Rapid Evolution of Model Properties

Although viewpoints provide an invaluable mechanism for managing disparate concepts within a design, these views are not without interactions. Typically, it is the designers' responsibility to maintain consistency among views. Furthermore, as the system is evolved, maintaining this consistency is a non-trivial task. Model aspect weaving technology offers a mechanism to automate these interactions.

The Constraint-Specification Aspect Weaver (C-SAW) is a model transformation engine that unites the ideas of aspect-oriented software development (AOSD) [18, 20, 31] with MIC to provide better modularization of model properties that are crosscutting throughout multiple layers of a model [12, 13]. C-SAW is available as a GME plug-in and provides the ability to explore numerous modeling scenarios by considering crosscutting modeling concerns as aspects that can be inserted and removed rapidly from a model [12].

Within the C-SAW infrastructure, the language used to specify model transformation rules and strategies is the Embedded Constraint Language (ECL), which is an extension of OCL. The ECL provides many of the common features of OCL, such as arithmetic operators, logical operators, and numerous operators on collections (e.g. size, forAll, exists, select). It also provides special operators to support model aggregates (e.g. models, atoms, attributes), connections (e.g., source, destination) and transformations (e.g. addModel, setAttribute, removeModel) that provide access to modeling concepts that are within the GME. There are two kinds of ECL specifications: a modeling pointcut describes the binding and parameterization of strategies to specific entities in a model; and a strategy is used to specify elements of computation and the application of specific properties to the model entities. C-SAW interprets these specifications and transforms the input source model into the output target model. The C-SAW web site is a repository for downloading papers, software, and several video demonstrations that illustrate model transformation with C-SAW in the GME (please see: http://www.gray-area/org/Research/C-SAW). The following sections provide representative examples of ECL and the concept of aspect model weaving.

4.1 Weaving Across Finite State Machines

Typically, when writing a specification for QoS adaptation, there arises one dimension of the adaptation that is treated as a dependent variable. There are numerous other independent variables that are adjusted to adapt the dependent variable according to some QoS requirement. For example, the end-to-end latency of video stream distribution may be a dependent variable that drives the adaptation of other independent variables (e.g., the size of a video frame, or even the video frame rate). In such cases, a *policy* is defined that represents the process for performing QoS adaptation. *The actions prescribed by the policy often crosscut the structure of a hierarchical state machine*. Changing the policy requires modifying each location of the state machine that is affected by the policy. Elrad et al. have also reported on scenarios where state machine models are crosscutting [8], but their approach is notational in nature and does not utilize a weaver at the modeling level.

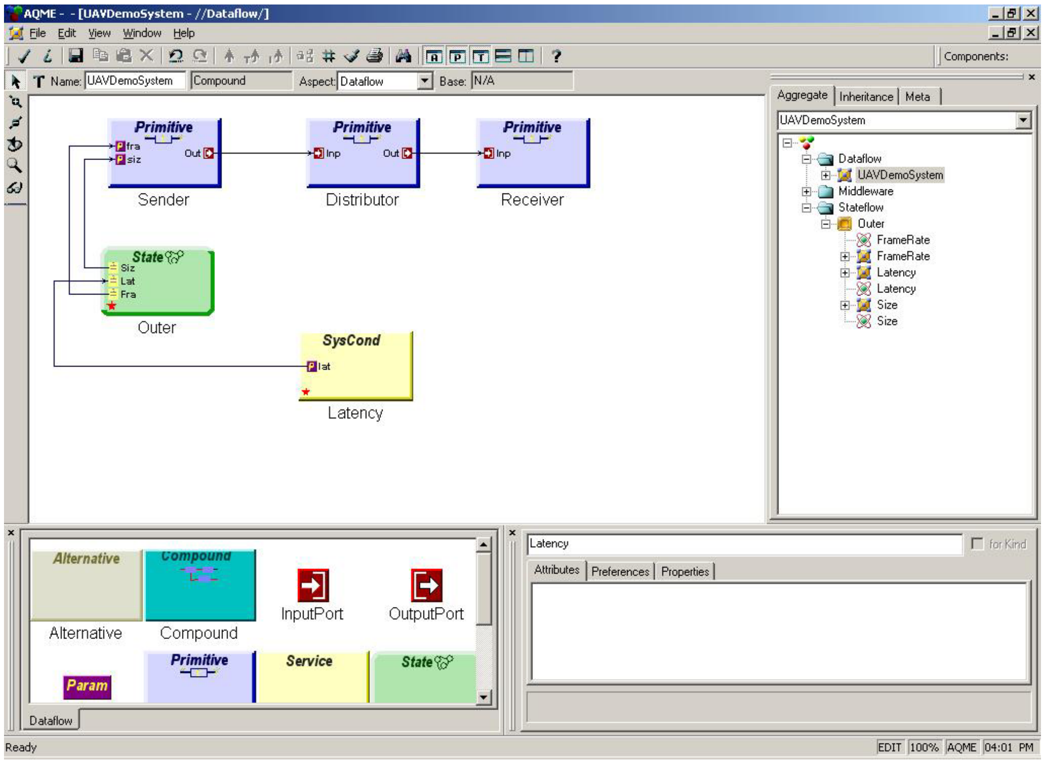


Figure 10: Dataflow for UAV Prototype

The C-SAW weaver has been applied to the AQML paradigm to assist in weaving properties according to the semantics of an adaptation policy. Several strategies have been created to support the modeling of state machines that represent the behavior of a contract. The first strategy focused on issues related to the creation of state machines and their internal transitions. The view of the model shown in Figure 10 pertains to the dataflow of the UAV case study. This model is an instance of the AQML metamodel. The latency concern is the dependent variable in this case. It represents a system condition object (the value of a system condition object is monitored from the environment). The latency is an input into a hierarchical state machine called "Outer." Within Outer, there are internal state machines that describe the adaptation of identified independent control variables (e.g., FrameRate and Size, as shown with the dependent Latency variable in Figure 11).

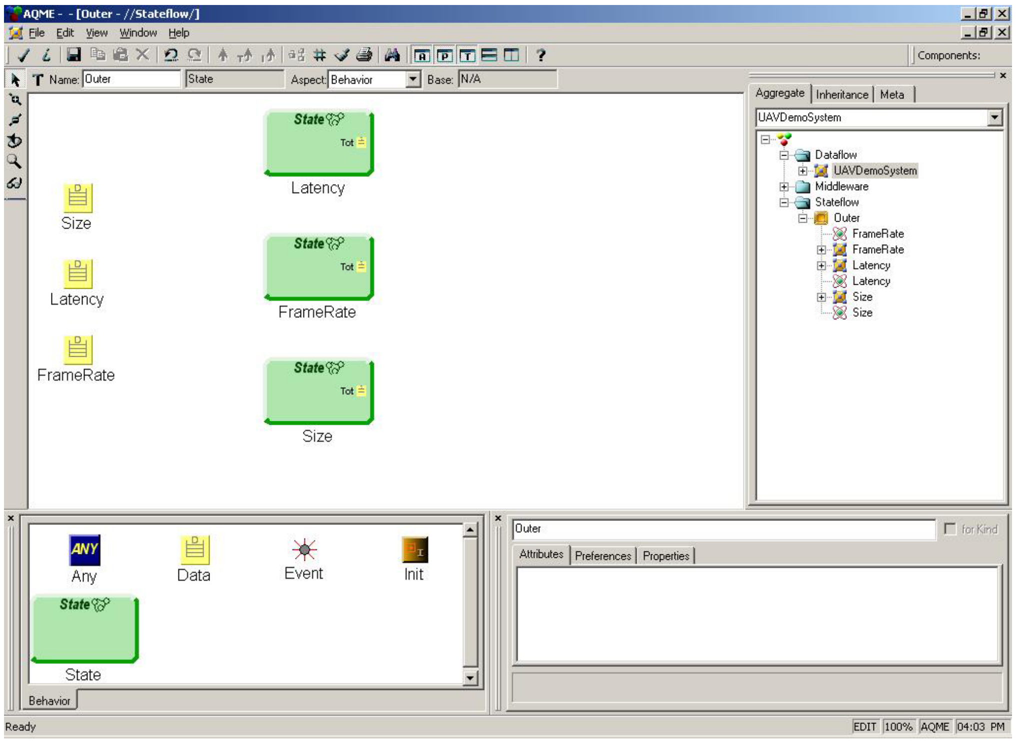
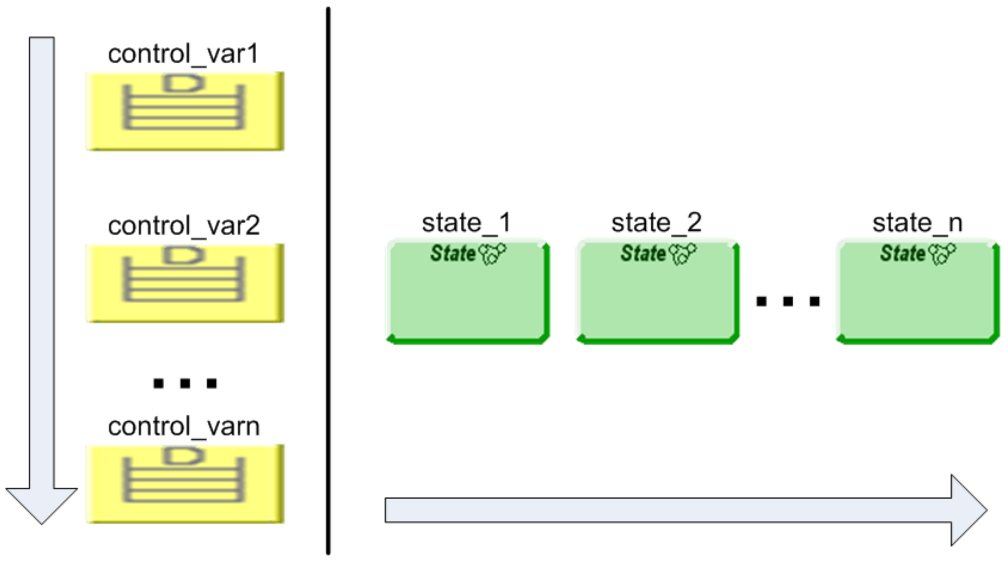


Figure 11: Top-Most View of Parallel State Machine

As depicted in Figure 12, there are two ways that a state machine model can be extended. Along one axis of extension, the addition of new dependent control variables often can offer more flexibility in adaptation toward the satisfaction of QoS parameters. It could be the case that other variables in addition to FrameRate and Size would help in reducing the latency (e.g., color, video format, compression). Figure 12a captures the intent of this extension through the introduction of new control variables. It may also be the case that, within a particular state, finer granularity of the intermediate transitions would permit better adaptation to QoS requirements. Figure 12b captures the intent of this extension.

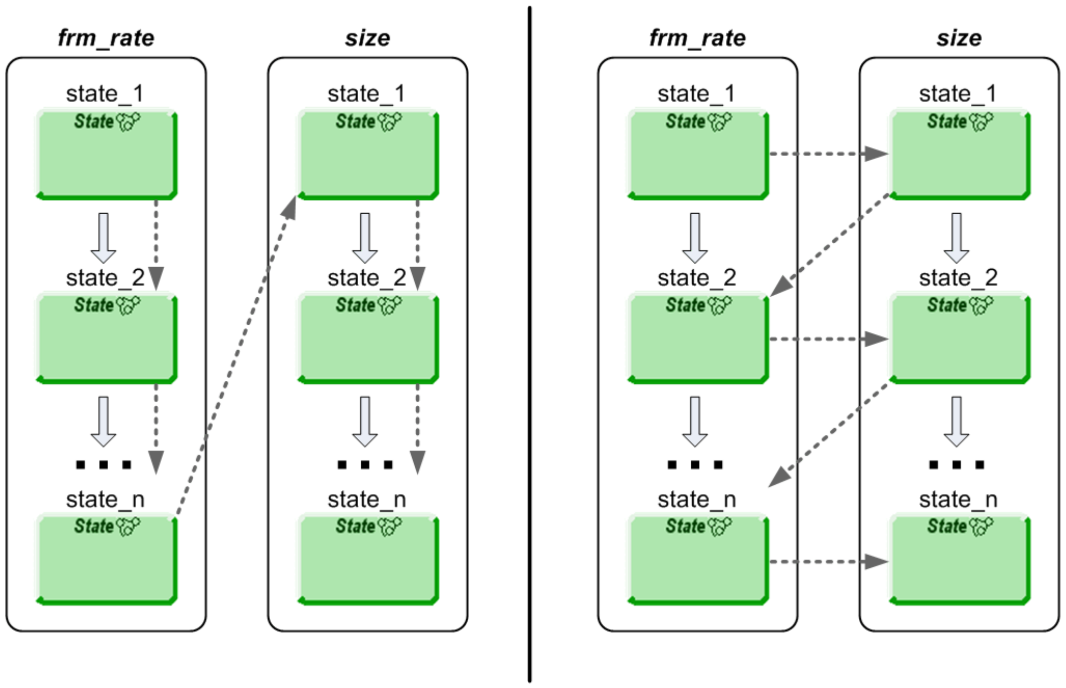


a) Adding New Control Variables   b) Adding More Intermediate Transitions in States

Figure 12: Axes of Variation within a State Machine

In addition to the strategy for creating control variables (and their intermediate states), an additional strategy was written to provide assistance in changing the adaptive policy that spans across each state machine. There could be numerous valid policies for adapting a system to meet QoS requirements. Two possibilities are given in Figure 13. The realization that each of these policies is *scattered across the boundaries of each participating state machine* suggests that these protocols represent a type of crosscutting concern that should be separated in order to provide an ability to change the policy rapidly.

The left-hand side of Figure 13 specifies a protocol that exhausts the effect of one independent variable (frm_rate) before attempting to adjust another independent variable (size). The semantics of this protocol pertain to the exhaustive reduction of one variable before attempting to reduce another one. Thus, the size variable is of a higher priority in this case because it is not reduced until there is no further reduction possible to the frame rate. The dotted-arrow in this figure indicates the order in which the transitions fire based upon the predicate guards.



a) Priority Exhaustive b) Zig-zag

Figure 13: Policies for Adapting to Environment

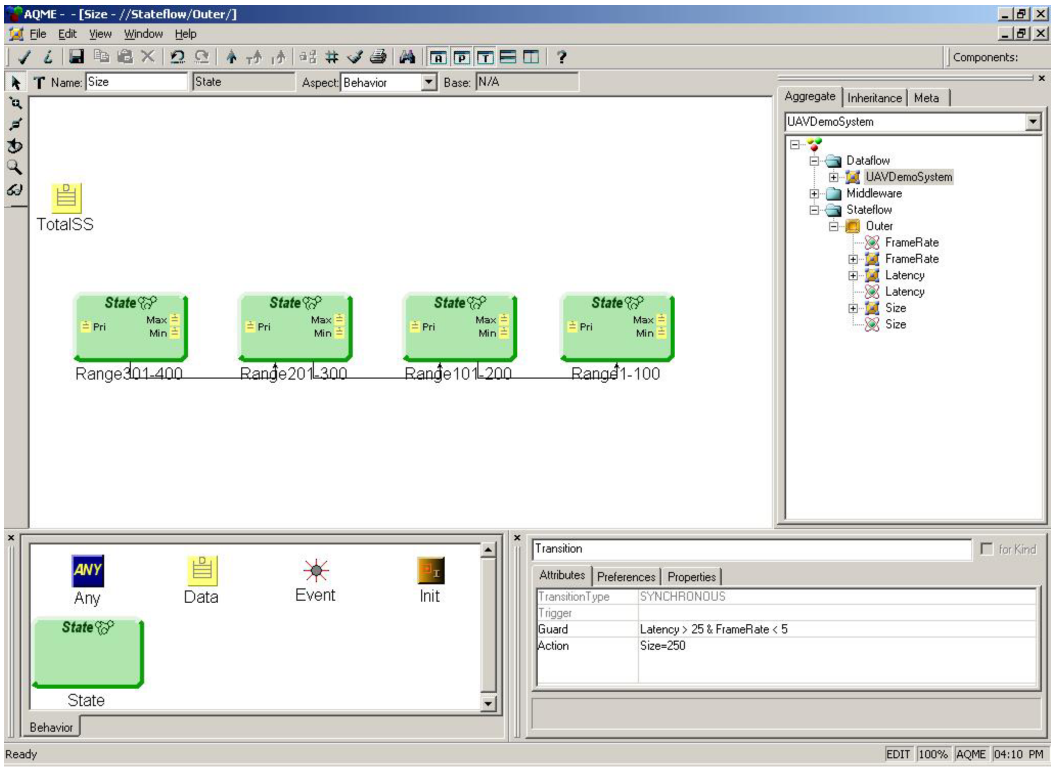
The right-hand side of Figure 13 represents a more equitable strategy for maintaining the latency QoS requirement. In this protocol, a zig-zag pattern suggests that the reduction of a variable is staggered with the reduction of a peer variable. Observe that the figure above involves only two control variables. The ability to change the protocol (by hand) becomes complicated when many variables are involved, or when there are numerous intermediate states. This crosscutting nature suggests that a strategy would be beneficial to assist in the exploration of alternative policies. Figure 14 contains a strategy that supports the protocol highlighted above in Figure 13a.

There may be several different variables that can be the focus of adaptation, depending on the contract and goals of an application. In the scenario specified in the strategy of Figure 14, a smaller frame rate is tolerated in order to maintain a desired latency. The transitions between two sub-states with different priorities will be inserted by the strategies specified in Figure 14. The strategy "*Apply-Transitions*" in line 50 retrieves the *stateName*, finds out the corresponding sub-state model, and calls another strategy "*FindConnectingState*" (line 32) that will then retrieve the priority of this sub-state. If the current priority is less than 4, another strategy "*AddTransition*" will be invoked in order to insert the transition from the current sub-state to the next sub-state with a higher priority. Line 12 through Line 14 find out the next sub-state whose priority is just 1 higher than the current state. Line 16 through Line 26 is the implementation for the insertion of the transition, along with associated attributes.

```
defines AddTransition, FindConnectingState, ApplyTransitions;

strategy AddTransition(stateName, guard : string;
                       prev: object; prevPri : integer)
{

  declare pri, minVal, maxVal, avgVal : integer;
  declare end : object;
  declare aConnection : object;
  declare action : string;

  pri:= findAtom("Priority").getIntAttribute("InitialValue");

  if(pri == prevPri + 1) then

    end := self;
    minVal := findAtom("Min").getIntAttribute("InitialValue");
    maxVal := findAtom("Max").getIntAttribute("InitialValue");
    avgVal := (minVal + maxVal) / 2;

    action := stateName;
    action := action + "=" + intToString(avgVal);

    aConnection := parent().addConnection("Transition", end, prev);
    aConnection.addAttribute("Guard", guard);
    aConnection.addAttribute("Action", action);

  endif;

}

strategy FindConnectingState(stateName, guard : string)
{

  declare pri : integer;
  declare start : object;

  pri:= findAtom("Priority").getIntAttribute("InitialValue");
  start := self;

  if(pri < 4) then

    parent().models("State")->
                  AddTransition(stateName, guard, start, pri);

  endif;

}

strategy ApplyTransitions(stateName, guard : string)
{

  declare theModel : object;

  theModel := findModel(stateName);
  theModel.models("State")->FindConnectingState(stateName, guard);

}
```

Figure 14: Latency Adaptation Transition Strategy

As a result, the weaving of the above strategy into the model of Figure 11 produces the internal view of the “size” state, shown in Figure 15. Each of the states progressively reduces the size of the video frame. The guard condition for the selected transition appears in the lower-right hand side of the figure. The guard condition states that the transition fires when the latency is not at the desired level, and also when the frame rate has been reduced to its smallest possible size.



Name: Size | State | Aspect: Behavior | Base: N/A

Transition

TransitionType	SYNCHRONOUS
Trigger	
Guard	Latency > 25 & FrameRate < 5
Action	Size=250

Figure 15: Internal Transitions within the Size State

4.2 Scaling a Base Model to Include Multiple UAV-s

In this scenario, all of the UAV-s must share bandwidth and communicate with a ground station that is responsible for allocating bandwidth share to the UAV-s according to their operational modes. The behavior of the ground station, as well as its interaction with each UAV, is modeled in AQML as well. It is a relatively modest effort for a user to develop a scenario involving two UAV-s that are interacting with one ground station. However, a challenge arises when the scenario needs to be scaled up to real-world sizes involving several UAV-s (100’s) and several Ground Stations (10’s). The models and the synthesis tools available in AQML may assist in mitigating the complexity to some extent by synthesizing a large fraction of the software; however, a significant share of complexity would be transferred over to the modeler who would need to build models of the larger real-world scenario.

Consider the single-UAV model of Figure 7 from Section 3. The difficulty of scaling a model to include additional UAV-s is rooted in the implicit relationships between a UAV and the modeled control adaptation. The *control logic to perform the requisite adaptation is scattered across numerous entities* such that the addition of a new UAV necessitates changes to many other locations in order to attach the UAV into the model. The model-weaver technique applies exceedingly well to this situation. The general solution separates all of the complexities of interaction into a strategy that can be reused in many design choices in order to scale to larger numbers of UAV-s.

```
1    defines Start, addUAV_r, addUAV, updateDF;
2
3    aspect Start()
4    {
5      addUAV_r(3,2);
6    }
7
8    strategy addUAV_r(max,idx: integer)
9    {
10     if (idx <= max) then
11       addUAV(idx);
12       addUAV_r(max,idx+1);
13     endif;
14   }
15
16   strategy addUAV(idx: integer)
17   {
18   rootFolder().findFolder("Dataflow").findModel("UAVDemoSystem").updateDF(idx);
19   }
20
21   strategy updateDF(idx:integer)
22   {
23       //The declaration of the local variables are omitted here
24
25       id_str := intToString(idx);
26       uav_csm := rootFolder().findFolder("Stateflow").findModel("UAV_CSM");
27       uav_ins := addInstance("State", "UAV_LC" + id_str, uav_csm);
28
29       tnr := rootFolder().findFolder("Middleware").findModel("TNR");
30       itnr_ins := addInstance("SysCond", "ITNR" + id_str, tnr);
31       otnr_ins := addInstance("SysCond", "OTNR" + id_str, tnr);
32
33       tc := rootFolder().findFolder("Middleware").findModel("TC");
34       tc_ins := addInstance("SysCond", "TC" + id_str, tc);
35
36       sdr := findModel("SDRPipe");
37       sdr_ins := addInstance("Compound", "SDRPipe" + id_str, sdr);
38
39       uins_itnr := uav_ins.findAtom("imag_tnr");
40       itins_tnr := itnr_ins.findAtom("TNR");
41       addConnection("Control", uins_itnr, itins_tnr);
42       addConnection("Control", itins_tnr, uins_itnr);
43
44       // The creations of other connections between the components are omitted
45   }
```

Figure 16: ECL Strategies for Replication of UAV-s

Figure 16 shows the ECL strategies for inserting two more UAV-s into the modeled system that was originally described in Figure 7. The aspect "*Start*" (Line 3) is the initiation point of the model weaving process. The strategy "*addUAV_r*" (Line 8) recursively invokes the strategy "*addUAV*"(Line 16) that will then call the strategy "*updateDF*" (Line 21) in order to replicate the UAV and its interactions with the other components in the system. The UAV reference is created from Line 25 to Line 27, which is followed by the creation of the related components such as "*TNR*" (Lines 29 to Line 31), "*TC*" (Lines 33 and 34), "*SDRPipe*" (Lines 36 and 37), as well as the connections between the UAV and "*TNR*" (Line 39 to Line 42). Many other connections are performed in the actual strategy, but they have been omitted here for the sake of brevity (the addition of other interactions follow similarly to those specified here).

As a result, Figure 17 illustrates the evolved system with three interacting UAV-s. This model can be compared to the original single-UAV model of Figure 7. Without the capability to separate the intent of UAV replication, the ability to scale a model to multiple UAV-s presents a manual modeling task that is too burdensome in practice. The use of C-SAW to automate the process permits rapid exploration of a base-line model in order to explore alternative scenarios involving multiple UAV-s. The concept of adding multiple ground stations is not shown here, but would be a similar burden if left solely to manual adaptation of a model.

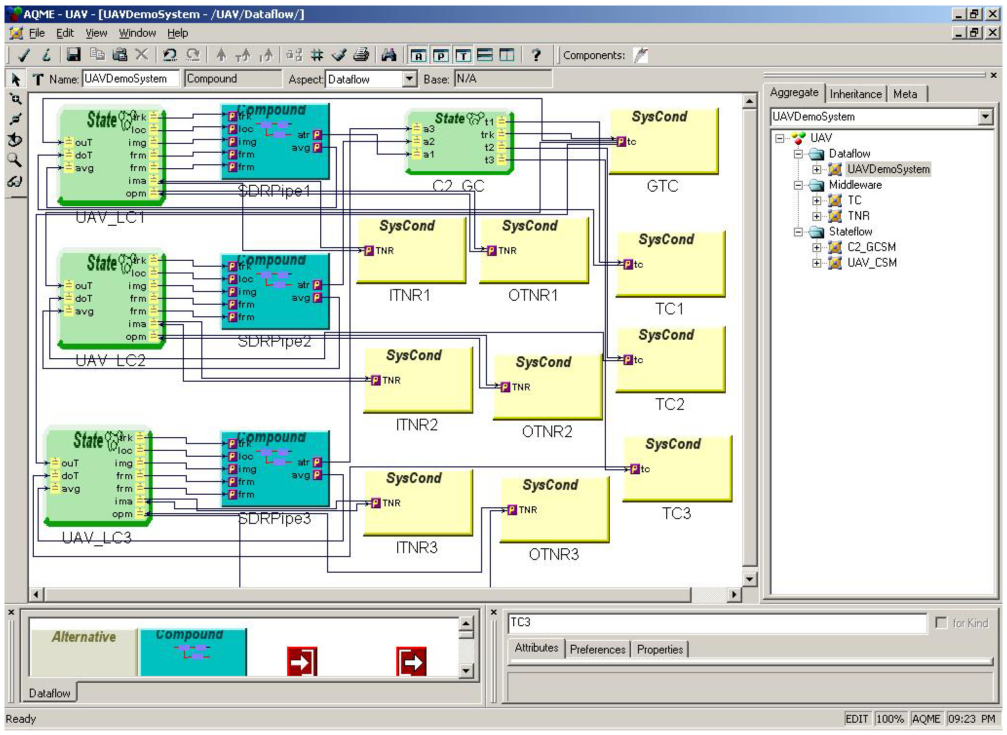


Figure 17: System with 3 UAV-s after Weaving

4.3 Synthesis of AQML Models for Simulation and Execution

Using a model-based representation and employing generators to synthesize low-level artifacts from models successfully raises the level of abstraction, and provides better tool support for analyzing the adaptation software. Our research has led to an approach that synthesizes adaptive contract descriptions from models, which permits the creation of larger and more complex contracts than could have been specified manually. A GME modeling paradigm has been created that synthesizes state-machine models into CDL contracts, and also Matlab simulation scripts. Initial results suggest an increase in productivity, due to: 1) shortening of the design, implement, test, and iterate cycle by providing early simulation, and analysis capabilities, and 2) facilitating change maintenance: *minimal changes in the weaving of properties into models can make large and consistent changes in the lower-level specifications* that would not scale well using manual reconfiguration of the models. Details about the generation process from AQML is described in [22]. The following two subsections provide a summary of the artifacts that are generated from AQML.

4.3.1 Matlab Simulation Generator

One of the primary goals of our approach is to be able to provide integration with tools that can analyze the QoS adaptation from a control-centric viewpoint. Matlab Simulink/Stateflow® is an extremely popular commercial tool that is routinely used by control engineers to design and simulate discrete controllers. Simulink provides the ability to model hybrid systems (mixed continuous and discrete dynamics) in a block-diagrammatic notation. Stateflow provides the ability to model hierarchical parallel state machines in a Statechart-like notation [14]. A Stateflow model can be inserted in a Simulink model as a block and the Simulink blocks can provide input stimulus and receive outputs from the Stateflow block. The Simulink/Stateflow model can be simulated within the Matlab framework for a desired time period, which in effect steps through the state-machine for the given input excitation. The responses from the state machine can be graphically plotted and the trajectory of the state machine can be visually observed, as well as recorded, for post-analysis. Additional analyses are possible in terms of the time spent in different states, the latency from the time of a change in excitation, to the time of change in outputs in the state machine, and stability of the system. Thus, the Matlab Simulink/Stateflow tool-suite provides an extremely convenient and intuitive framework for observing and verifying the behavior of the system.

We have implemented a generator, using our model interpretation technology, which can translate the QoS adaptation specifications (modeled using the AQML) into a Simulink/Stateflow model. Matlab provides an API that is available in the Matlab scripting language (M-file), for procedurally creating and manipulating Simulink/Stateflow models. The simulation generator produces an M-file that uses the API to create Simulink/Stateflow models. Many details of synthesis to Matlab are documented in [22].

4.3.2 CDL Generator

Research at BBN on DRE systems and QoS adaptation resulted in the Contract Definition Language (CDL) – a domain-specific language (based on the OMG Interface Definition Language) for specifying QoS contracts. BBN has produced a CDL compiler and a QoS adaptation kernel that can process specifications (contracts) expressed in CDL. Additional aspect languages have also been created by BBN to support the adaptation effort [7]. The CDL compiler translates QoS contracts into artifacts that can execute the adaptation specifications at run-time. CDL is a textual language, and it has a state-machine like flavor (see [27] for details). The modeling efforts described in this paper build upon the BBN infrastructure to affect the adaptation specifications captured in the AQML models. That is, CDL specifications are generated from the AQML models, and the BBN tools are used to instantiate these adaptation instructions at runtime in a real system. In [22], results are presented that illustrate the situation where a simple state change within an AQML model translates into dozens of changes that would be needed at the CDL level. This suggests that QoS adaptation can be modeled in a manner that is more scalable than an equivalent process using a textual language.

5. Conclusion

Composition of artifacts across the software lifecycle has been the focus of several recent research efforts [1, 24, 30]. The synergistic power resulting from a combination of modeling and aspects enables changes to be made rapidly to a high-level system specification, which can be synthesized into a simulation or implementation [13]. This paper introduced an approach based on MIC for simulating and generating QoS adaptation software for DRE systems using modeling aspects and viewpoints. The key focus of the approach is on raising the level of abstraction in representing QoS adaptation policies, and providing a control-centric design for the representation and analysis of the adaptation software. This approach has similar goals to MDA [2, 10], as well as AOSD [18, 20, 31] (but applied across multiple software artifacts, similar to [1, 24]).

The approach has been tested and demonstrated on a UAV Video Streaming application described as a case study in this paper (NOTE TO REVIEWER: if accepted, we plan to have video demonstrations of this specific application available on our web page by the time of publication). The case study from Section 4 presented a simple scenario; however, C-SAW has enabled the modeling of a much larger scenario in the developed modeling environment, with a higher degree of variability and adaptability. In our experience, the simulation capabilities have been particularly helpful in fine-tuning the adaptation mechanism. In addition to the Unmanned Aerial Vehicle (UAV) case study presented in this paper, the GME and C-SAW have been applied also to Bold Stroke [25] - a mission avionics computing platform from Boeing (please see [13] for C-SAW usage on GME models representing Bold Stroke event channels).

The literature on aspect modeling with the UML has the most direct relation to the work described in this paper. A growing body of literature is forming around the topic of aspect modeling, and is perhaps best chronicled by the annual workshops on Aspect-Oriented Modeling (AOM) (see http://www.cs.iit.edu/~oaldawud/AOM/). A representative selection of papers in the AOM area can be found in [4, 5, 15, 28]. A focal point of these efforts is the development of notational conventions that assist in the documentation of concerns that crosscut a design. These notational conventions advance the efficiency of expression of these concerns in the model. Moreover, they also have the important trait of improving the traceability from design to implementation. Although these current efforts do well to improve the cognizance of AOSD at the design level, they generally tend to treat the concept of aspect-oriented modeling primarily as a specification convention. This is to say that the focus has been on the graphical representation, semantical underpinnings, and decorative attributes concerned with aspects and their representation within UML. A contribution of this paper is to consider AOM more as an operational task by constructing executable model weavers. That is, we view AOSD as a mechanism to improve the modeling task, itself, by providing the ability to quantify [9] properties across a model *during* the system modeling process. This action is performed by utilizing the C-SAW weaver that has been constructed with the concepts of domain-specific system modeling in mind.

There are several enhancements that have been identified as future areas for investigation. We plan to integrate a symbolic model-checking tool (e.g., SMV [3]) for formal reasoning about the adaptation mechanism. With the aid of this tool, various properties can be established (such as liveness, safety, and reachability) about the state-machine implementing the adaptation policy. We also plan to strengthen the computation and middleware modeling in order to facilitate analysis of the application and middleware components. The topic of testing model transformation is another area that will be explored in the near future. As model transformations become a large part of the model-driven process, it is desirable to have a testing framework that can be used to verify the correctness of each transformation rule. We are currently developing a testing framework that resides within the GME to execute and compare model transformations with C-SAW.

A summary of the benefits of concern-driven modeling are illustrated in Figure 18. A viewpoint is an important concern separation technique that allows different perspectives to be modeled through partitioning mechanisms provided by the GME tool. With viewpoints, attention can be focused on related segments of a model without being overwhelmed by details unrelated to the concern of interest. From the base model, an aspect model weaver can be used to perform global transformations that span the hierarchy of the base model. Aspects at the modeling level enable various design alternatives (e.g., properties related to QoS specification) to be explored rapidly. From a transformed model, techniques from generative programming [6] can be applied to synthesize the model into a form suitable for simulation or model checking. After verification of the models, code can be generated that captures the execution semantics of the modeled system (along with the specified QoS). The iterative nature of the process enables the evaluation of numerous system configurations before actual deployment.

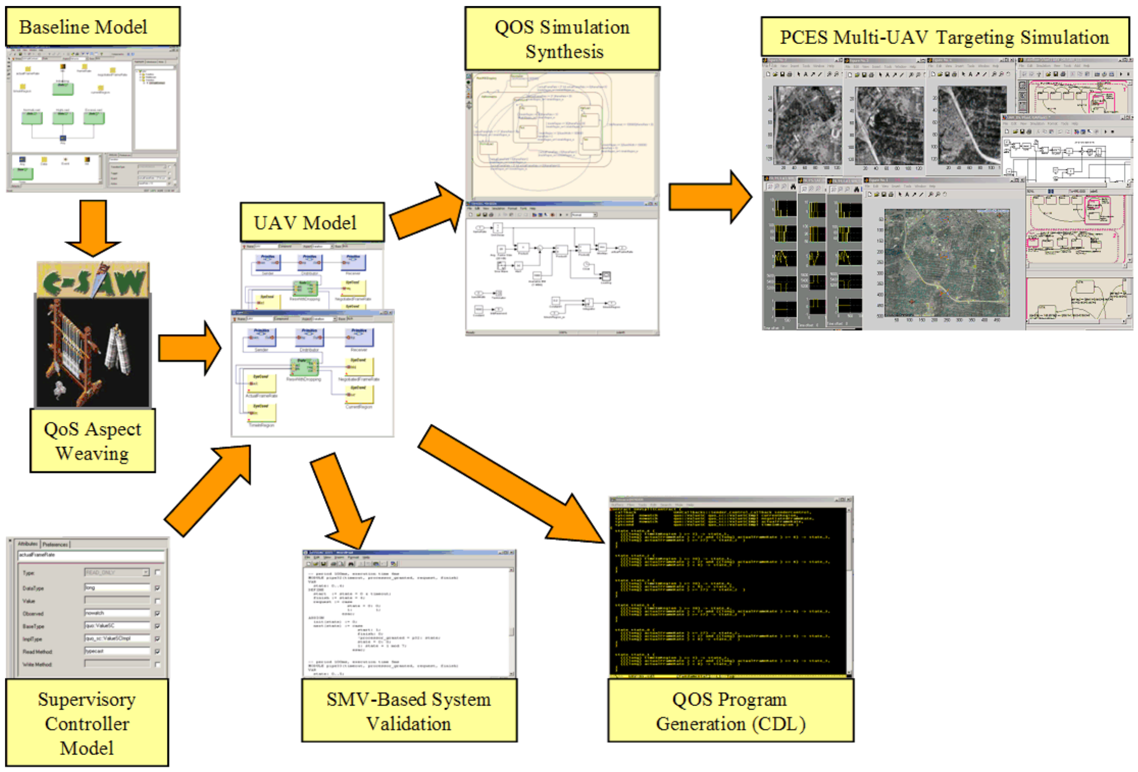

Baseline Model
QOS Simulation Synthesis
PCES Multi-UAV Targeting Simulation
UAV Model
C-SAW
QoS Aspect Weaving
Supervisory Controller Model
SMV-Based System Validation
QOS Program Generation (CDL)

Figure 18: Summary of Concern-Driven QoS Modeling

Acknowledgments

This work is supported by DARPA under the Program Composition of Embedded System (PCES) program within the Information Exploitation office (DARPA/IXO). The authors also acknowledge Joseph Loyall and Richard Schantz of BBN Technologies, and Craig Rodrigues, for valuable discussions and comments.

References

1. Don Batory, Jacob Neal Sarvela, and Axel Rauschmayer, “Scaling Stepwise Refinement,” *IEEE Transactions on Software Engineering*, accepted for publication – Fall 2004.
2. Jean Bézivin, “MDA: From Hype to Hope, and Reality,” *The 6th International Conference on the Unified Modeling Language*, San Francisco, California, Keynote talk, October 22, 2003.
3. Jerry Burch, Edmund Clarke, Kenneth McMillan, David Dill, and L. J. Hwang, “Symbolic model checking: 10^20 states and beyond,” *Information and Computation*, vol. 98, no. 2, June 1992, pp. 142--70.

4. Tony Clark, Andy Evans, and Stuart Kent, “Aspect-oriented Metamodelling,” *The Computer Journal*, 46(5), September 2003, 566-577.
5. Siobhán Clarke, “Extending standard UML with model composition semantics,” *Science of Computer Programming*, 44(1), July 2002, pp. 71-100.
6. Krzysztof Czarnecki and Ulrich Eisenecker, *Generative Programming: Methods, Tools, and Applications*, Addison-Wesley, 2000.
7. Gary Duzan, Joseph Loyall, Richard Schantz, Richard Shapiro, and John Zinky, “Building Adaptive Distributed Applications with Middleware and Aspects,” *AOSD '04: International Conference on Aspect-Oriented Software Development*, Lancaster, UK, March 22-26, 2004, pp. 66-73.
8. Tzilla Elrad, Omar Aldawud, and Atef Bader, “Aspect-Oriented Modeling: Bridging the Gap between Modeling and Design,” *Generative Programming and Component Engineering (GPCE)*, LNCS 2487, Pittsburgh, Pennsylvania, October 2002, pp. 189-201.
9. Robert Filman and Daniel Friedman, “Aspect-Oriented Programming is Quantification and Obliviousness,” *OOPSLA Workshop on Advanced Separation of Concerns*, Minneapolis, Minnesota, October 2000.
10. David Frankel, *Model Driven Architecture: Applying MDA to Enterprise Computing*, Wiley Publishing, 2003.
11. Aniruddha Gokhale, Douglas Schmidt, Balachandran Natarajan, Jeff Gray, and Nanbor Wang, “Model-Driven Middleware,” in *Middleware for Communications*, (Qusay Mahmoud, editor), John Wiley and Sons, 2004.
12. Jeff Gray, Ted Bapty, Sandeep Neema, and James Tuck, “Handling Crosscutting Constraints in Domain-Specific Modeling,” *Communications of the ACM*, Oct. 2001, pp. 87-93.
13. Jeff Gray, Janos Sztipanovits, Douglas C. Schmidt, Ted Bapty, Sandeep Neema, and Aniruddha Gokhale, “Two-level Aspect Weaving to Support Evolution of Model-Driven Synthesis,” in *Aspect-Oriented Software Development*, (Robert Filman, Tzilla Elrad, Mehmet Aksit, and Siobhán Clarke, eds.), Chapter 29, Addison-Wesley, 2004.
14. David Harel, “Statecharts: A Visual Formalism for Complex Systems,” *Science of Computer Programming*, June 1987, pp. 231-274.
15. Mohamed Kande, and Valentin Crettaz, “Towards Patterns for Concern-Oriented Software Architecture,” *The 4th AOSD Modeling With UML Workshop*, San Francisco, CA, October 2003.
16. David Karr, Craig Rodrigues, Joseph Loyall, Richard Schantz, Yamuna Krishnamurthy, Irfan Pyarali, and Douglas Schmidt, “Application of the QuO Quality-of-Service Framework to a Distributed Video Application,” *International Symposium on Distributed Objects and Applications*, Rome, Italy, September 2001, pp. 299-309.
17. Gábor Karsai, Miklos Maroti, Ákos Lédeczi, Jeff Gray, and Janos Sztipanovits, “Type Hierarchies and Composition in Modeling and Meta-Modeling Languages,” *IEEE Trans. on Control*

System Technology (special issue on *Computer Automated Multi-Paradigm Modeling*), March 2004, pp. 263-278.

18. Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, and John Irwin, "Aspect-Oriented Programming," *European Conference on Object-Oriented Programming (ECOOP)*, LNCS 1241, Springer-Verlag, Jyväskylä, Finland, June 1997, pp. 220-242.
19. Ákos Lédeczi, Arpad Bakay, Miklos Maroti, Peter Volgyesi, Greg Nordstrom, Jonathan Sprinkle, and Gábor Karsai, "Composing Domain-Specific Design Environments," *IEEE Computer*, November 2001, pp. 44-51.
20. Karl Lieberherr, Doug Orleans, and Johan Ovlinger, "Aspect-Oriented Programming with Adaptive Methods," *Communications of the ACM*, October 2001, pp. 39-41.
21. Joseph Loyall, Richard Schantz, John Zinky, Partha Pal, Richard Shapiro, Craig Rodrigues, Michael Atighetchi, David Karr, Jeanna Gossett, and Christopher Gill, "Comparing and Contrasting Adaptive Middleware Support in Wide-Area and Embedded Distributed Object Applications," *IEEE International Conference on Distributed Computing Systems (ICDCS-21)*, Phoenix, Arizona, April 2001, pp. 625-634.
22. Sandeep Neema, Ted Bapty, Jeff Gray, and Aniruddha Gokhale, "Generators for Synthesis of QoS Adaptation in Distributed Real-Time Embedded Systems," *Generative Programming and Component Engineering (GPCE)*, LNCS 2487, Pittsburgh, Pennsylvania, October 2002, pp. 236-251.
23. Basher Nuseibeh, Jeff Kramer, and Anthony Finkelstein, "A Framework for Expressing the Relationship Between Multiple Views in Requirements Specification," *IEEE Transactions on Software Engineering,* October 1994, pp. 760-773.
24. Awais Rashid, Ana Moreira, and João Araújo, "Modularization and Composition of Aspectual Requirements," *2nd International Conference on Aspect-Oriented Software Development*, Boston, Massachusetts, March 2003, pp. 11-20.
25. Wendy Roll, "Towards Model-Based and CCM-Based Applications for Real-Time Systems," *Proceedings Sixth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC '03)*, Hokkaido, Japan, May 14-16, 2003, pp. 75-82.
26. Richard E. Schantz and Douglas C. Schmidt, "Middleware for Distributed Systems: Evolving the Common Structure for Network-centric Applications," *Encyclopedia of Software Engineering*, Editors John Marciniak and George Telecki, Wiley and Sons, New York, 2001.
27. Richard Schantz, Joseph Loyall, Michael Atighetchi, and Partha Pal, "Packaging Quality of Service Control Behaviors for Reuse," *The 5th IEEE Symposium on Object-oriented Real-time distributed Computing (ISORC)*, April 2002, Washington, DC, pp. 375-385.

28. Dominik Stein, Stefan Hanenberg, and Rainer Unland, "Issues on Representing Crosscutting Features," *Third International Workshop on Aspect-Oriented Modeling with UML*, Boston, March 18, 2003.
29. Janos Sztipanovits and Gábor Karsai, "Model-Integrated Computing," *IEEE Computer*, April 1997, pp. 10-12.
30. Peri Tarr, "Toward a More Piece-ful World," *Keynote at Generative Programming and Component Engineering (GPCE),* September 22-25, 2003, Erfurt, Germany, pp. 265-266.
31. Peri Tarr, Harold Ossher, William Harrison, and Stanley Sutton, "N Degrees of Separation: Multi-Dimensional Separation of Concerns," *International Conference on Software Engineering (ICSE)*, Los Angeles, California, May 1999, pp. 107-119.