MODEL-BASED FRAMEWORK TO DESIGN QoS ADAPTIVE DRE APPLICATIONS

By

Sujata Mujumdar

Thesis

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

MASTER OF SCIENCE

in

Computer Science

December, 2005

Nashville, Tennessee

Approved:

Dr. Gabor Karsai

Dr. Sandeep Neema

`

To,

My Family

ACKNOWLEDGEMENTS

This research was sponsored by the MoBIES Program of Defense Advanced Research Projects Agency (DARPA)/IXO, under Contract number #F33615-C-02 -1285.

I would like to extend my deepest gratitude towards my Research Advisor, Dr. Sandeep Neema, for providing me with an opportunity to work on this project. His direction and guidance throughout the development of the project has been extremely helpful. I would also like to thank Dr. Gabor Karsai for providing me with insightful comments regarding the work done towards this thesis and for providing valuable suggestions to improve it.

I wish to express my sincere appreciation to Nagabhushan Mahadevan who has, in the initial phases of the project, helped me to develop certain aspects of the work. Additionally, I wish to thank all the members and friends in ISIS who have provided me great company when I needed a break from work.

My parents have always been my source of inspiration. They have provided me with boundless encouragement for pursuing my goals and have given me meaningful education. I would like to thank them for this precious gift. I would also like to thank my in-laws for all their blessings and my husband for providing support. His belief in me and constantly helping me to keep the right perspective on the important things has been extremely helpful. Last, but definitely not the least, I want to thank my friend, Sayali, for constantly checking up on the status of my work and for providing motivation to complete it on time.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

DRE	Distributed Real-Time and Embedded
RT-CORBA	Real -Time – Common Object Request Broker Architecture
RMI	Remote Method Invocation
COM	Component Object Model
QoS	Quality-of-Service
QuO	Quality Objects
CDL	Contract Definition Layer
IDL	Interface Definition Layer
OMG	Object Management Group
MIC	Model-Integrated Computing
GME	Generic Modeling Environment
OLC	Online Control
DSME	Domain Specific Modeling Environment
COTS	Component-off-the-shelf
IP	Internet Protocol
DARPA	Defense Advanced Research Projects Agency
SM	Service Manager
HSM	Higher-level Service Manager
AQML	Adaptive QoS Modeling Language
FCS	Feedback Control Real-Time Scheduling
UML	Unified Modeling Language

DQME	Dynamic QoS Modeling Environment
MBC	Model Based Controller
CC	Composite Controller
SRML	Software Radio Modeling Language
FSM	Finite State Machine
UAV	Unmanned Aerial Vehicle
RUAV	Reconnaissance Unmanned Aerial Vehicle
UCAV	Unmanned Combat Aerial Vehicle
CAOC	Combined Air and Operations Control
AOI	Area Of Interest
PCES	Program Composition and Embedded Systems
CPU	Central Processing Unit

CHAPTER I

INTRODUCTION

Performance-critical distributed systems operating in unpredictable environments have been proliferating in the past decades [1]. Such systems are often classified as distributed real-time and embedded (DRE) systems, and are finding increasing application in military and defense context, flight avionics, industrial process automation, among others. Rapid advances in middleware technologies like RT-CORBA [15], COM+ [16] and Java RMI [17], have been primary drivers in increasing applications of DRE systems.

Designing and implementing DRE systems however, is significantly challenging when compared to the development of traditional non-embedded systems due to a variety of factors. The real-time reactive nature of the driving application, inherent distribution of the application software components over a set of potentially resource constrained host nodes, communication over a potentially unreliable and most likely unpredictable network, unpredictability of the environment, as well as unforeseeable variability in the workload, are all factors that interact in intractable ways to enhance the design complexity. The design is specifically challenging, since despite of the unpredictability and unreliability of the resources, infrastructure, and the environment, the overall application is still subject to the most stringent reliability and predictability requirements – for example, in a mission computing avionics application which is responsible for weapons release, an untimely response to the pilot's weapon's release command could potentially result in catastrophe.

An important aspect of large-scale, mission critical DRE systems therefore, is the necessity to guarantee a certain level of Quality-of-Service (QoS) for e.g. ensuring minimum guarantees on the system performance in terms of parameters like latency, throughput, jitter, security and reliability (for description of these terms, see Appendix B). In the context of DRE systems, QoS implies an adherence to a set of quality requirements or properties in order to optimize the performance of the applications [75]. QoS adaptation involves taking actions to tune the application parameters in order to achieve a desired QoS level. The set of parameters used to tune and observe a DRE application are defined as QoS Parameters. QoS parameters are used to provide certain guarantees pertaining to the performance of an application. These guarantees are termed as QoS Guarantees [74].

Meeting stringent QoS requirements while taking into account a multitude of factors, such as those listed above, in real-time, for a distributed application operating in dynamic conditions is a complex task. However, ensuring QoS guarantees is crucial for mission-critical applications; for e.g., one may need to guarantee the CPU utilization on multiple processors to be able to meet end-to-end deadlines [2]. Failure to meet the QoS guarantees may result in mission failures and potentially severe physical and financial damage among other consequences [3].

The specification and implementation of QoS in DRE systems has typically been done using ad-hoc schemes [13] [14] and using weak abstractions to adaptively meet the design requirements. These however, are not systematic enough to ensure system integrity or reusability. The ad-hoc schemes employed require highly skilled professionals who, based on experience, use intuition and domain knowledge devising ways to tune the applications in order to achieve the desired QoS behavior. This dependence on expertise, intuition, and ad-hoc techniques makes it impossible to formally guarantee the performance, while prohibiting a widespread adoption of the QoS adaptation technique.

Prior researches sought to improve the state-of-the-art in QoS provisioning by introducing a QoS adaptation layer on top of the middleware [68]. The rationale is to achieve a separation of concerns between the QoS and the functional aspects of the DRE systems by providing a mechanism to specify the QoS requirements and adaptation policies orthogonally from the application design and implementation. The QuO [12] project at BBN developed a QoS enabled middleware, which incorporates the adaptation layer, and includes software infrastructure to specify and enforce the QoS adaptation policies. QuO provides a Contract Definition Language (CDL) which is an extension of OMG's Interface Definition Language (IDL) that allows describing QoS adaptive behavior of the objects and object interactions, and specifies QoS guarantees as Contracts. While this does advance the state-of-the-art from the perspective of separation of concerns and enhanced reusability, the approach still suffers from a degree of ad-hocism. In CDL, the QoS adaptations are specified as modifications (or "tuning") to QoS parameters in response to change in system operating conditions. However, these parameter "tunings" are again based on expertise and intuition, rather than being grounded in any mathematical or formal theory. It is worth noting that the DRE systems and the applications implemented atop these systems belong, from a mathematical point of view, to a class of highly complex non-linear dynamical systems with intractable couplings and feedbacks. Adaptation of the system in an ad-hoc manner without regards to the dynamics may potentially result in unstable behaviors. Moreover, the CDL offers a code-centric low-level of abstraction, which is difficult to analyze and cumbersome to manage.

The realization that a DRE system could be viewed as a complex dynamical system, at least from the QoS perspectives, opens up the possibility of applying feedback control techniques for adaptively managing the QoS requirements. The general approach in this class of techniques involves abstracting the DRE system as a physical plant with a set of sensing and actuation interfaces, designing the QoS adaptation logic as a controller, and setting up the sensing, control, and actuation feedback loop. Unfortunately, designing the control logic has been the prerogative of control engineers who make use of Control Theory [4] to design effective feedback loops, which makes the technique not easily amenable to DRE systems and software engineers.

Control theoretic approaches have proved to be successful in a number of physical plant control systems [5] [6] [7] [8]. The basic premise of control theory is runtime feedback control even if accurate models of the system to be controlled are not available. Many applications have made use of the control-theoretic approaches in improving the performance of the applications. However, the control-engineers, at times, make assumptions of the functional details of the systems. The domain engineer specializes in the functional aspects of the system, unlike control engineers. Nevertheless, the domain engineer is not suited to design mechanisms for QoS adaptations in these systems due to a lack of control-theoretic background. Consequently, a framework that can capture the domain knowledge while abstracting the control knowledge for use by domain engineers is needed.

Model Integrated Computing (MIC) [11] provides a way for designing and implementing such a framework, with a domain level of abstraction. GME [9] [10], a meta-programmable environment, developed at Institute for Software Integrated Systems (ISIS), Vanderbilt University is based on the methodology of MIC and has been refined over years for the creation and synthesis of complex computer-based systems. GME has been used in various domains [18] [19] [20] [21] [22] [23]. Domain specific models can be constructed to capture the domain knowledge and provide solutions to complex problems of designing large scale DRE systems. Modeling enables the representation of systems in terms of their environments, platforms, design of the system along with the implementation details at an abstract level. Modeling various parameters enables the domain engineers to anticipate the consequences of multiple integrations. The paradox of the increasing complexity during the development of embedded systems along with reducing the time and cost for development can be resolved by use of modeling paradigms and their underlying support such as the frameworks for analysis of models and code synthesis.

Problem Statement

This thesis proposes a model-based framework that provides a higher level of abstraction to the domain engineers enabling an effective representation of QoS design problems and adaptation strategies which have traditionally been the prerogative of control-engineers. The proposed model-based framework shall provide an ability to effectively capture the interactions of different QoS variables, as opposed to using ad-hoc or intuitive policies. It shall further enable domain engineers to develop functional aspects of the DRE application in a coherent manner to ensure the seamless integration with QoS adaptations mechanisms, while at the same time separating the functionality from QoS. The framework will facilitate a systematic development of design time methodologies to develop the run-time adaptations, simplifying the development of dynamically-adaptive construction of model interpreters is also proposed for code synthesis. This code will represent the internal logic of the designed controllers and may be used in the low-level implementation frameworks.

CHAPTER II

BACKGROUND AND LITERATURE REVIEW

This chapter reviews and presents a background on the state-of-the-art technologies used for building QoS in DRE systems. It is divided in three sections: The first section gives a background on the various technologies that are relevant in the context of the work presented in this document. The second section discusses the existing approaches in the context of DRE systems and control-theoretic approaches. The third section gives an overview of the tools that have been used towards the implementation of the work presented.

Background

Quality-Of-Service (QoS)

As discussed earlier, supporting the needs of complex large-scale DRE systems requires QoS management that is adaptive to the dynamism in the environment. Construction of a QoS framework within a system involves mapping of QoS requirements to resources, QoS specifications that capture the QoS requirements of the application and QoS adaptation mechanisms which realize desired QoS behavior [31].

QoS Specification

Specification of QoS can be done at various system levels for e.g., protocol layers like transport/network, middleware or other applications. QoS Specification includes specifying requirements for performance, synchronization, QoS management, cost and the level of service. Expected performance characteristics are needed to establish resource commitments. Specification of synchronization includes characterizing the degree of synchronization between related services or events. Specification of the level of service for QoS states the degree of resource commitment required to maintain performance guarantees. The cost of service signifies the price a user is willing to incur to obtain a desired level of service. QoS management is the degree of QoS adaptation that can be tolerated and scaling actions to be taken in the event the contracted QoS cannot be met.

QoS requirements are specified by the high-level parameters of an application that convey what the user requires. An assessment of the QoS requirements should be performed to determine if they can be met. In case the specified level of service cannot be provided then trade-offs need to be specified.

QoS Mapping

The various resource requirements that can be derived from the QoS requirements need to be mapped onto quantitative QoS parameters. Mapping enables the monitoring and control of the system parameters. QoS parameters may be oriented towards performance, format or cost of the service among others. Each QoS parameter can be viewed as a typed variable with bounded values and the values are subject to negotiation between the system layers.

QoS Enforcement

Resource management must be QoS-driven in order to provide and sustain QoS. The resource management system must not only consider resource availability and resource control policies, but also an application's QoS requirements measured in terms of the QoS parameters when allocating resources. Resources should be re-allocated in response to variations in the system environment. All the participating elements should negotiate collectively for ensuring that the QoS parameters will be satisfied. Negotiation involves the dynamic adaptation, transmission and translation of QoS parameters between the various participating layers. After resources are allocated, the resource manager components are responsible to guarantee the sustained availability of the allocated resources. This also requires monitoring of the resource availability and its dynamic characteristics. If a change in the system leads to degradation in the QoS and the resource manager cannot make appropriate resource adjustments, then the application can either adapt to the new level of QoS or scale to a reduced level of service.

Essential elements for QoS Identification

QoS characteristics define a space in which an application has unacceptable quality of service when the values of any of the QoS parameters fall 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.

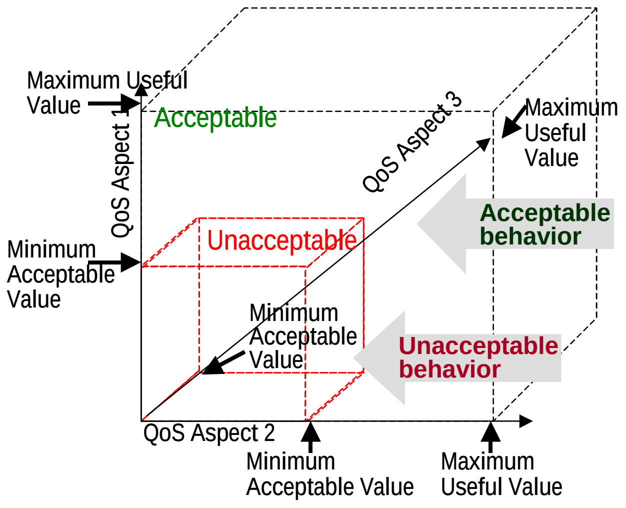


Figure 1 QoS characteristics define an operating space

QoS dimensions of interest are changeable and they vary with the number of QoS parameters that are relevant to the application. For simplication, Figure 1 illustrates only three such dimensions; however the space can consist of any number of dimensions corresponding to the QoS parameters relevant to the application. Moving from one point in this QoS space to another may not occur smoothly. The underlying system, mechanisms, and resource managers provide knobs to control the level of QoS that also define the granularity. Adaptation of one QoS dimension may affect other QoS dimensions. Finally, the application's requirements will often mean that some tradeoffs and adaptations are preferred over others. Five essential questions are posed when trying to build a distributed system with support for dynamic adaptation in QoS space:

- What does the system need to achieve?
- Where is the system currently within the QoS space?
- Where can the system go within the QoS space?
- How can the system go from one point to another in the QoS space?

- Where should the system move to in the QoS space?

These questions are addressed in the proposed model-based framework (discussed in Chapter 3) developed for attaining and maintaining QoS in distributed real-time and embedded systems.

Control Theory

A traditional way of achieving service level objectives in complex systems is by adding a controller to a system in order to tune the system parameters. There are various methodologies that make use of classical control theory [4] to build such controllers. Application of classical control theory involves system identification that requires mathematical models of the system to be built and designing the controllers to tune the system parameters.

Control theory relies on the foundations of runtime feedback control to monitor and influence the behavior of dynamic systems. A dynamic system is one that changes states in response to various external environmental conditions or configuration changes over time. Classical feedback control (Figure 2) consists of a feedback loop that connects the application/plant to a controller in a closed-loop scheme. The objective here is to keep the plant's output close to a reference value.

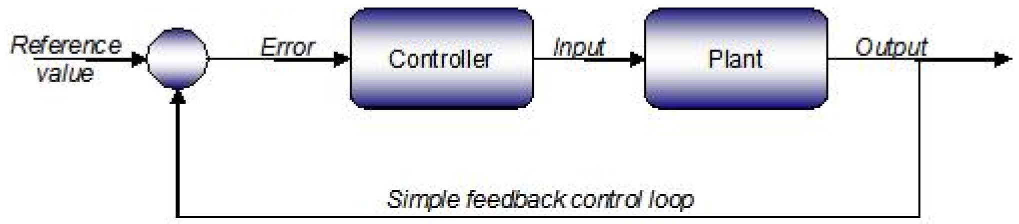


Figure 2 Feedback Control Loop

Based on the adaptation strategies, the controller is required to modify the plant's input parameters in order to achieve the desired system behavior. The desired behavior is specified using state or reference variables. In control theory, a controller is often responsible to control the outputs or states of the dynamic system usually achieved using a feedback control loop as seen in Figure 2. Systems utilizing feedback are also known as closed-loop control systems. The feedback is used to make decisions about the changes to the control signals that drive the plant or the application.

An open-loop control system, on the other hand, doesn't have or use the feedback control [29]. Systems having an open-loop controller need to be well-characterized so as to predict the inputs to achieve the desired outputs thus making feedback unnecessary. There may be situations in which an open-loop control system may be preferred over closed-loop ones. An example may be of a system in which periodic resetting is possible so as to eliminate or at-least suitably control the effects of initial errors and later disturbances [30]. However, sometimes the lack of connections between the outputs of the system to the inputs may cause poor performance of the system under control and therefore the open-loop controller may be undesirable.

Control theory offers a systematic way to design automated and efficient resource management schemes. Computer systems hosting applications critical to military command

and control, commerce and banking among others need to obey strict QoS requirements while operating in highly dynamic environments. To achieve the desired QoS, numerous performance-related parameters must be continuously optimized to respond rapidly to changing computing demands. The current state-of-the-art in designing these systems to meet their QoS requirements involves substantial manual intervention as will be seen in next section. If the computer system of interest is correctly modeled and the effects of its operating environment accurately estimated, control algorithms can be developed to achieve the desired performance objectives.

Online Control Approach (OLC)

Utilizing control theory for resource management requires the establishment of an appropriate model of the underlying systems dynamics. This model is responsible to capture the relationship between the observed system parameters and the control inputs used for adjusting various parameters. An initial approximate model may be built for those system components whose dynamics are known and parameter estimation techniques can be used to identify the unknown parameters of the system [80]. Abdelwahed et. al. have developed a model predictive Online Control (OLC) [78] approach in which control actions are derived to optimize system behavior for pre-specified QoS criteria over a limited look-ahead prediction horizon. This type of model-predictive control method is more powerful and more widely applicable to computational systems, which may be highly non-linear than simple feedback control. The OLC approach allows control objectives to be represented explicitly in the form of multi-variable optimization problem that is solved at every control step. The OLC can control processes with varied characteristics from those with relatively simple dynamics to ones with complex dynamics, as well as systems with long delay or dead times.

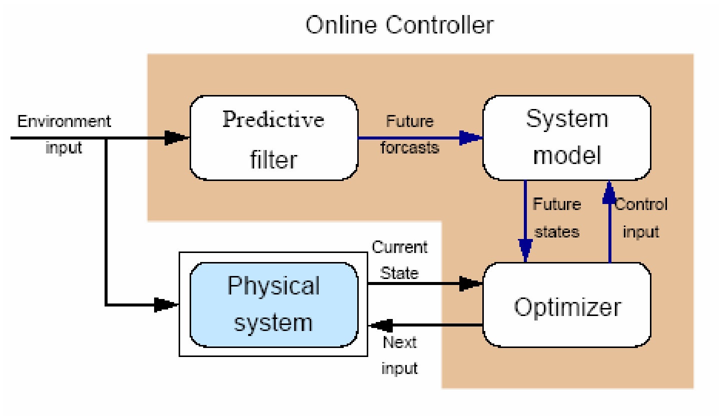


Figure 3 Online Control Structure [78]

Figure 3 shows a generic online controller. Relevant parameters of the operating environment are such as data arrival patterns are estimated and used by the system model to forecast the future behavior over a look-ahead horizon [80]. The online controller principally aims to satisfy the desired QoS requirements (set-point specifications) by continuously monitoring the current system state and selecting the control inputs that best satisfy them. The controller explores only a limited forward horizon in the system's state space.

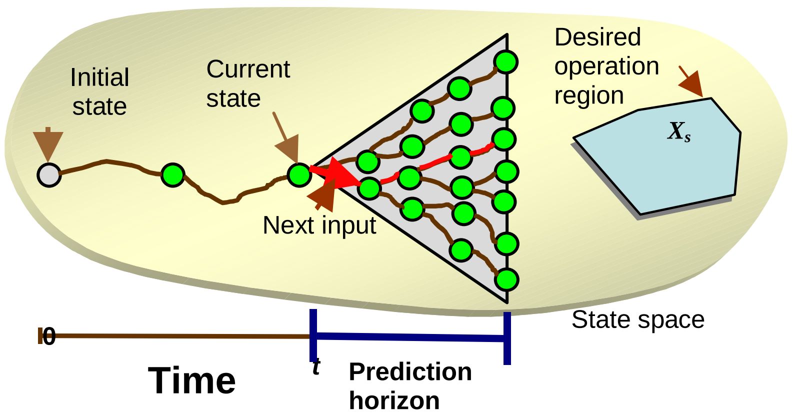


Figure 4 Limited look-ahead horizon approach [80]

As seen in Figure 4, at every time step it constructs a (limited) tree of all possible future states given all possible control inputs, and selects the trajectory that reduces a cost-function while satisfying constraints. The input that leads to this trajectory is chosen as the next control input to the system. This process is repeated at each time step. This control policy takes into account the effects of possible variations in environment inputs.

System Dynamics Representation

Capturing system dynamics is one of the most important and generally one of the most difficult of all tasks. Static systems that generate an output on a certain input are easier to model as at any point in time, the output is always dependent on the instantaneous value of the inputs to the system. However, in case of dynamic systems, the output of a system is dependent not only on the current inputs of the system but also on the past inputs [71]. Differential equations can be used to express the relation between the inputs and the outputs in dynamic systems. In a general case of describing the system with differential equations, higher order derivatives of the output variables can be described as functions of lower order derivatives of the output variables and some derivatives of the input variables.

In the work presented, system dynamics have been approached using State-space representation of the system.

State-Space Representation

A description of the system dynamics can be obtained using the state-space representation also known as time-domain approach. It provides the dynamics as a set of coupled first-order differential equations in a set of internal variables known as *state variables*, together with a set of algebraic equations that combine the state variables into physical output variables [72]. The concept of the *state* of a dynamic system refers to a minimum set of state variables that fully describe the system and its response to any given set of inputs. However, for complex system the set of state variables is not unique. This means that for a system with certain inputs, outputs and 'n' state variables, there exists infinite number of state-space representations for the system. For each value of time t, a state that lies in n-dimensional state space can be obtained [68].

MIC

Model-Integrated Computing (MIC) [11] [26] developed at Institute for Software Integrated Systems (ISIS), Vanderbilt University, is a technology that addresses the problems of developing software integrated systems by providing rich, domain-specific modeling environments (DSME). Domain-Specific models (DSM) focus on high level abstractions of the problem space, avoiding low-level details. MIC is used to create multi-aspect (See Appendix A.5) models to facilitate systems engineering analysis of the models and to automatically synthesize applications from the models. Multi-aspect modeling, in MIC, allows capturing of all relevant system information in various aspects. A significant application of MIC is in systems that have a tight integration between the physical configuration and the computational structure of the system. MIC has proved to be a powerful tool by providing adaptability in changing environments [27].

Existing Approaches

This section presents an overview of the applications using various techniques to achieve a desired degree of QoS.

COTS Middleware

Middleware, as defined in computer science, consists of software agents that act as intermediaries bridging the gap between different application components. One of the most common instances of middleware is software that resides between the applications and the underlying operating systems and networks. It is responsible to bridge the functionality gaps between the applications and the hardware/software infrastructure [32] [33]. Top-level applications are programmed to target the middleware, which abstracts out the OS and/or hardware-specific functionality, thus providing a common interface for interaction with different OS' and/or hardware to the high-level components.

Component middleware, originally designed to support enterprise systems, has been applied extensively to DRE systems. Component middleware is a category of middleware that allows component services to be composed, configured and deployed to create robust DRE systems [34]. Middleware technologies associated with components provide standardized, off-the-shelf component interconnecting solutions supporting better reuse of software across product families. The commercial-off-the-shelf (COTS) component middleware is used to develop high-performance real-time embedded systems.

Examples of COTS middleware include: Object Management Group’s (OMG) CORBA [35] and Real-time CORBA [15], Sun’s Jini [36], Java RMI [17], and EJB [37] frameworks, Microsoft’s DCOM [38], and IBM’s MQSeries message-oriented middleware (MOM) [39].

The objective of COTS middleware is to decrease the effort required to develop complex DRE systems by composing applications out of modular reusable software components and services rather than building them from scratch thus reducing the overall cycle-cost of development and testing. In the past few years, the COTS middleware has been used to develop scalable and robust large-scale DRE systems that have managed QoS requirements.

COTS middleware based on RT-CORBA specification allows the DRE applications to allocate, schedule and control the processor, memory and the networking resources. It thus allows the application developer to preserve operating system level priorities of schedulable tasks that can be distributed among various nodes in a real-time distributed system [40]. RT-CORBA based middleware address many QoS properties like defining policies that control connection multiplexing, request priority and queuing order [41]. However, the middleware based on RT-CORBA does not utilize technologies, like the ones developed by the networking community for ensuring QoS in IP, within the network to provide end-to-end QoS management. Additionally, the middleware also lacks strategies for transparent configuration of the QoS properties into the application thus having the application developers make the configuration decisions manually which tend to be error-prone.

To summarize, although usage of COTS middleware improves the reusability of software architectures, DRE systems require explicit interfaces and mechanisms for developing adaptive solutions. RTSJ [42] and Dynamic Scheduling RT-CORBA [43] are COTS middleware that provide some elements that can be used for the implementations of adaptive solutions. However, additional higher level approaches are needed that can realize easier specification of QoS in DRE systems. COTS middleware is not yet able to provide complete end-to-end solutions to support application development in diverse environments. Additionally, conventional COTS middleware does not enforce complex QoS requirements effectively as they were originally designed for applications with less stringent requirements.

Quorum Technologies

Exploration of adaptive techniques such as dynamic scheduling, reconfiguration and various resource management techniques is being conducted under the DARPA Quorum program [44]. Some of the projects developed under the Quorum program are described in this section.

BBN-QuO

Quality Objects (QuO) [12] [45] [46] [47] is a framework developed at BBN Technologies for providing QoS adaptation in network-centric distributed applications [48]. QuO adds QoS to CORBA and Java RMI in a manner which is appropriate for creating applications that can adapt to environments (that are unpredictable or have strict resource constraints). It has been used in a number of demonstrations and applications [49] ranging from wide-area distributed applications to embedded real-time systems. With QuO, distributed applications can specify

- their QoS requirements
- the system elements that must be monitored and controlled to measure and provide QoS
- the behavior for controlling and providing QoS and for adapting to QoS variations that occur at run-time.

QuO thus separates the role of functional application development from the role of developing the QoS behavior of the system. QuO middleware supports the role of *Qosketeers* [12] who are responsible for defining contracts, system condition objects, callback mechanisms, and object delegate behaviors. QuO provides an open source toolkit [50] [51] for supporting the development of QoS features. This toolkit contains a Quality Description Language (QDL) that describes QoS contracts and adaptive behaviors of object [51], code generators for weaving measurement, control, and adaptation code into application programs [50], a library of reusable system condition objects, a runtime kernel that coordinates evaluation of contracts and monitoring of system condition objects and an encapsulation model called Qoskets for encapsulating runtime behavior into reusable units.

To summarize, QuO provides a new modern technology enabling enhanced usability and separation of the functional from the QoS aspects of the DRE systems. However, it still relies on the fact that the QoS adaptations need to be controlled manually. Human operators are required to modify or tune the QoS parameters in response to changes in system conditions. These QoS adaptations, realized by tuning the QoS parameters, are heavily dependent on the expertise that the operator gains over time. This approach eventually suffers from a high-degree of ad-hocism rendering the systems with complex dynamics highly unstable.

DeSiDeRaTa

Dynamic, Scalable, Dependable, Real-Time (DeSiDeRaTa) [52] [53] project was developed to address adaptive resource management approach to enable reconfigurable ground and space information systems. It utilizes a *dynamic path paradigm*, which is employed for modeling and resource management of distributed real-time mission-critical systems. The dynamic path concept is used for automated QoS assessment and resource allocation. The path may be composed of sensors, actuators and control software for filtration, evaluation and action [54]. The primary metric used in DeSiDeRaTa is QoS that is a measure of path latency (end-to-end). Figure 5 shows the logical architecture of the DeSiDeRaTa QoS management software.

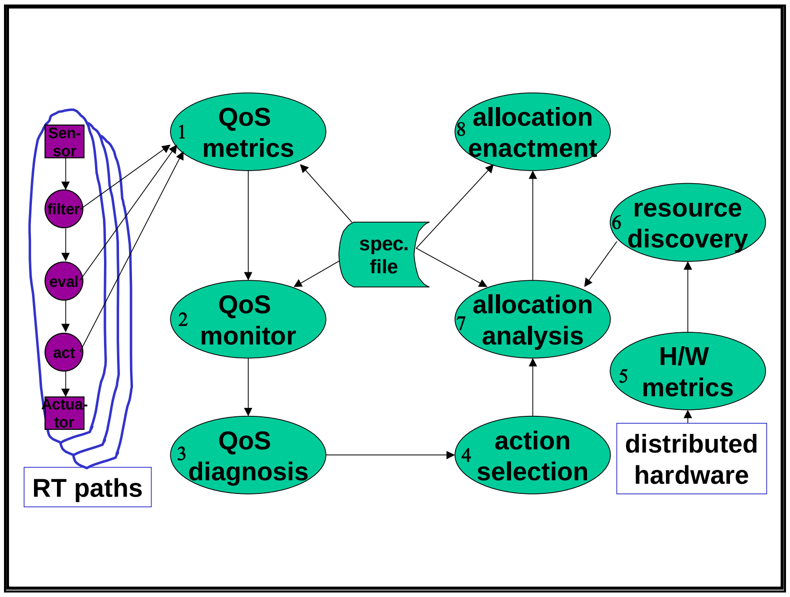


Figure 5 Logical architecture of the resource and QoS
management software [53]

The application programs of real-time control paths send time-stamped events to the *QoS metrics* component. The *QoS metrics* calculates path-level QoS metrics and sends them to the *QoS monitor*. The monitor checks if the observed QoS conforms to the required QoS and notifies the *QoS diagnosis* component when a QoS violation occurs. The diagnosing component notifies the *action selection* component of the cause(s) of poor QoS and recommends actions to improve QoS. Action selection ranks the recommended actions and forwards the results to the *allocation analysis* component; this component consults *resource discovery* for host and LAN load index metrics, and determines an efficient method for allocating the hardware resources to perform the actions, and requests that the actions be performed by the *allocation enactment* component. Path-level QoS specification is used by the adaptive resource allocator to determine if the current configuration is achieving the desired QoS and to assist in selecting new configurations to improve QoS.

Resource monitoring capabilities provide application profiles and instrumentation data. Application profiles include hardware resource requirements for particular QoS levels.

To summarize, the DeSiDeRaTa project offers a level of abstraction in terms of a *path* and provides for the monitoring and adaptive management of run-time QoS by taking decisions to dynamically (re)allocate resources as needed. However, DeSiDeRaTa is a middleware technology that does not offer separation between the functional aspects and QoS.

RT-ARM

The Real-Time Adaptive Resource Manager (RT-ARM) [55] developed by Honeywell Technologies is a reactive resource adaptation service. It is a middleware service that adapts the rate of tasks according to changing environmental conditions. The main components of RT-ARM are known as Service Managers (SM) [56]. The SMs are responsible for requesting, distributing and managing resources at different levels. The RT-ARM consists of various components like Client, Translator, Negotiator, Allocator, and Adapter.

A *Client* submits requests for QoS to the RT-ARM. Applications then start accessing the services and communicate with the resource management as a part of a process of changing QoS. The QoS Model in RT-ARM consists of three main parts: dimension, range and region. A QoS dimension is an aspect of a service that can be measured in a certain way e.g, frame rate, bandwidth are QoS dimensions. Each dimension has a QoS range (min, max). The pair (dimension, range) forms a QoS parameter. Sets of these pairs form a QoS region. The adaptation model consists of QoS shrinking/reduction, QoS expansion/improvement used when the resources either rise/drop and feedback adaptation.

The SMs have a hierarchical nature in which the higher level SMs (HSM) forms the root or nodes and the lower level SMs (LSM) form the leaf-nodes. The request submitted by the client is serviced by the HSMs. The client is thus saved from the details of lower-level parameters. The *negotiator* is responsible for processing the request when the client requests to negotiate a QoS contract. The *translator* performs the function of translating the higher level request by the client into dimensions and ranges understandable by the LSMs. The *allocator* allocates and releases resources when no QoS adjustments are needed. In case the QoS adjustments need to be performed, the *adapter* takes charge of allocating and releasing resources. When triggered to react, the RT-ARM manipulates the CPU usage of key operations. The RT-ARM performs this task by manipulating subset of task invocation event rates from application specified available rates. A QuO contract is responsible to prompt the RT-ARM to adjust the ranges of invocation rates to re-allocate more CPU cycles.

To summarize, the RT-ARM makes heavy use of middleware, CORBA, to obtain QoS. However, similar to the case of DeSiDeRaTa, RT-ARM does not provide a clear separation of concerns in terms of the functional and QoS aspects of the system.

AQML

Adaptive QoS Modeling Language (AQML) [68] is a precursor to the work presented in this thesis for building QoS adaptations in the DRE systems. It is a model-driven approach used for the creation of high-level graphical models. These models are created using AQML, a domain-specific modeling language, for representing the QoS adaptation strategies. It employs Stateflow [81] models to capture the adaptive QoS behavior of the system. A discrete FSM representation is used along with hierarchy and concurrency for modeling QoS adaptive behavior of the system. The states represent a discretized configuration of QoS properties. Transition objects are used for representing transitions from one state to another. Attributes are used for showing the decomposition of the states. Event and Data objects are modeled to capture Boolean event variables and the data objects are used to capture data. They can be scoped as being local to a state machine or an input/output variable of a state machine, by setting other attributes. Dataflow and functional components can be modeled in a different aspect of the same language. The coupling between QoS parameters and elements from the functional model can be done by inserting appropriate references.

To summarize, the AQML modeling language supports model-driven development of the QoS adaptive applications. However, the entire set of QoS properties are modeled solely as states in a state machine, and hence *discretized*. Moreover, expressing complex systems operating in dynamic conditions and exhibiting a continuous-time behavior in terms of only states is insufficient. A way to express the hybrid behavior or discrete-time and continuous-time realizations needs to be established. A way of separating concerns of QoS from the functionality of the systems also needs to be devised.

Control Theory based Applications

Control theory has been successfully applied to various systems involving uncertainties in the system models [57]. This section reviews some of existing applications making use of control-theoretic approaches to achieve QoS in the respective functionalities and concludes with a summary of the discussed applications.

Task Scheduling

Research conducted by Stankovic, Lu et. al. in [6] [58] makes use of control theory in scheduling tasks and maintaining QoS. A Feedback Control Real-Time Scheduling Architecture (FCS) has been built that is used for adaptive real-time systems. Control theory has been applied to design the FCS algorithms for satisfying the transient and steady-state performance specifications of real-time systems. The architecture developed allows one to plug in various real-time scheduling policies and QoS optimization algorithms. Figure 6 illustrates the application of the control theory to the FCS architecture.

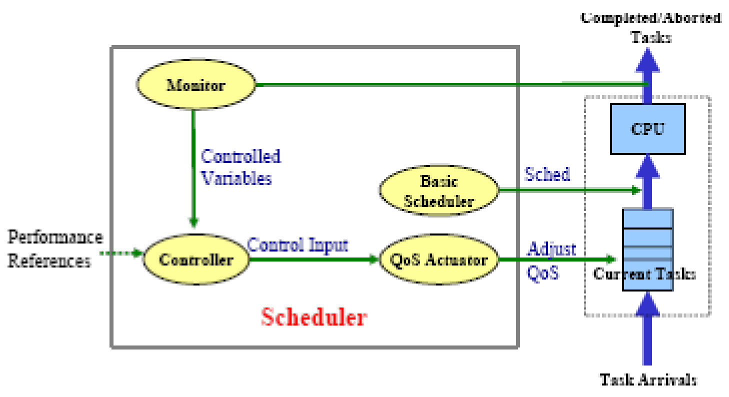


Figure 6 FCS architecture employing Control-Theory for feedback control [6]

The FCS architecture has a feedback control loop composed of a Monitor, a Controller, a QoS Actuator, and a Basic Scheduler. The scheduler is responsible to schedule tasks depending on the scheduling policy plugged in. Based on the analytical models, control theory is used to tune the controller and develop mathematical analyses of performance of the controller.

Bandwidth Allocation and QoS Adaptation in Web Servers

Abdelzaher, Bhatti in [7] [8] use a control-theoretic approach to provide QoS guarantees in a web-server. Some of the QoS guarantees provided are overload protection and performance-driven load-balancing. The work focuses on utilization control as a basic building block to achieve complex control objectives and to satisfy a wide range of performance requirements. They consider a client-server system in which clients send a succession of requests to the web server, each request having a deadline by which it must be served. The delay in the service of a request is dependent on the time taken in the network plus the time taken by the server. Their work concentrates on server-side delay as opposed to the int-serv [76] or the diff-serv [77] architectures that address the problem of network delays. The architecture makes an assumption of the web server facing single bottlenecks at any time and is geared towards serving static web content. They make use of a utilization controller in the feedback loop as seen in Figure 7.

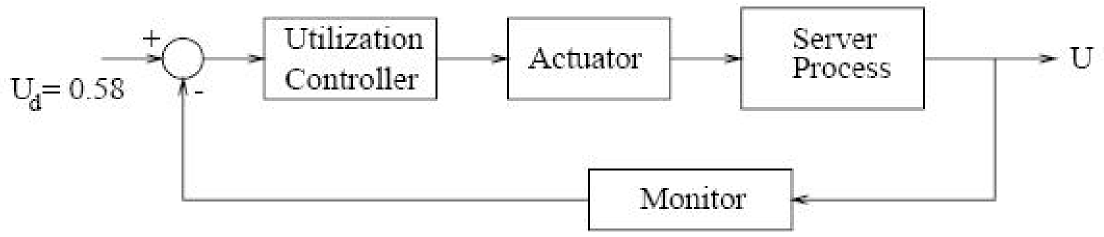


Figure 7 Web Server adaptation employing Control Theory in the feedback loop [7]

The control loop seen in the figure, measures the server utilization and based on the load conditions, determines a subset of clients that can receive service at that time. The size of the client subset is dynamic in the sense that only those many number of clients will be selected so as to keep the web server busy at all times and the utilization at the desired level.

Network Flow Control

Control theory presented in [62] is applied towards the application of congestion control in the network traffic. In this work, the dynamics of the network data queues in response to the input traffic is described using the control-theoretic approach. The following figure shows the feedback control loop as modeled for the network traffic.

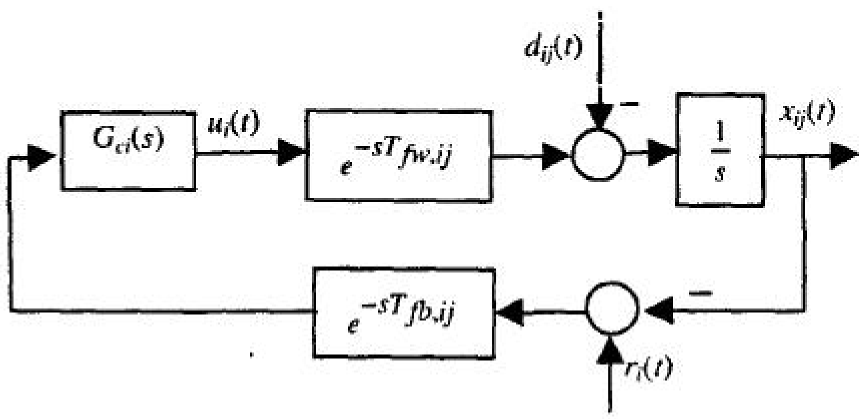


Figure 8 Model of the controlled TCP flow [62]

Figure 8 shows an integrator model (upper right box) that represents the buffer that is considered to be the bottleneck for the TCP flow. The disturbance models the bandwidth that is available for a flow at particular instant of time. The two transfer functions model the propagation time from the flow source to the bottleneck queue and from the bottleneck queue to the destination and then the source back. The Controller function has also been represented in terms of a transfer function. The feedback control scheme uses two inputs: the reference signal and the disturbance. Due to the possibly large propagation delays, the queue level dynamics might become unstable that needs to be controller. The design of the controller is based on the Smith principle [63][64]. Smith predictor is also known as a compensator for a stable process with large time delay. The controller thus designed guarantees the source input rate promptly utilizes all of the available bandwidth.

Power Management

The work described in [65], represents a formal feedback control algorithm for dynamic voltage/frequency scaling in a portable multimedia system. The main objective here is to save power while maintaining a desired playback rate. The feedback control system designed is shown in the Figure 9.

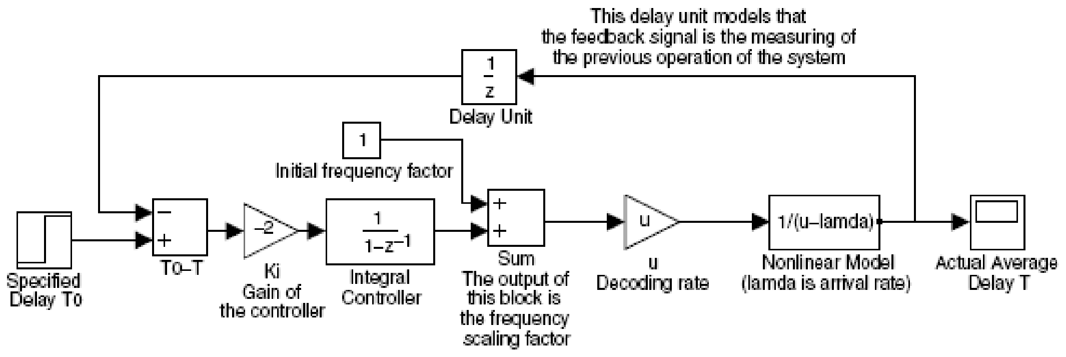

This delay unit models that the feedback signal is the measuring of the previous operation of the system
1/z
Delay Unit
1
Initial frequency factor
Specified Delay T0
T0–T
-2
Ki Gain of the controller
1/(1-z^-1)
Integral Controller
Sum The output of this block is the frequency scaling factor
u
u Decoding rate
1/(u–lamda)
Nonlinear Model (lamda is arrival rate)
Actual Average Delay T

Figure 9 Architecture of the designed control system [65]

At the beginning, the frequency scaling factor is unity i.e. the decoder is run at maximum speed. The initial value is forced by adding a constant "1" to the output of the controller. After the operation begins, the controller is responsible to give the current decision of the frequency scaling factor that will go to the decoder system by taking into consideration average frame delay under the previous frequency scaling conditions.

Control Theory based Applications - Conclusions

The systems discussed above make use of the control theory and classical feedback control to achieve the desired QoS behavior. They start by observing the current system state and subsequently taking corrective actions, if any are required, for attaining QoS. The success of these applications, of being able to attain certain QoS level, makes the control-theoretic approaches significantly advantageous. However, the applications discussed here usually assume a linearized and discrete-time model for system dynamics. In contrast, many practical systems exhibit hybrid behavior comprising both discrete-event and time-based dynamics [61] [66] [67]. Some of the applications discussed above require human intervention to input the reference variables to controllers to manage the trade-offs which makes it an extremely tedious process. Moreover, the QoS is integrated within the systems that make the techniques tightly coupled with the functionalities.

Developmental Tools

GME

Generic Modeling Environment (GME) [9] [10] is a domain specific, model-integrated program synthesis tool that implements the MIC technology. It is used for the creation of domain specific, multi-aspect models of large-scale engineering systems [11]. GME is configurable in the sense that it can be adapted to represent various application domains. This is achieved by defining Meta-models. Meta-models define the domain-specific elements of the modeling language. A graphical user-interface, based on UML notations [28], is provided in GME to facilitate the design of the system. Figure 10 gives an architectural picture of the way in which MIC has been applied in GME.

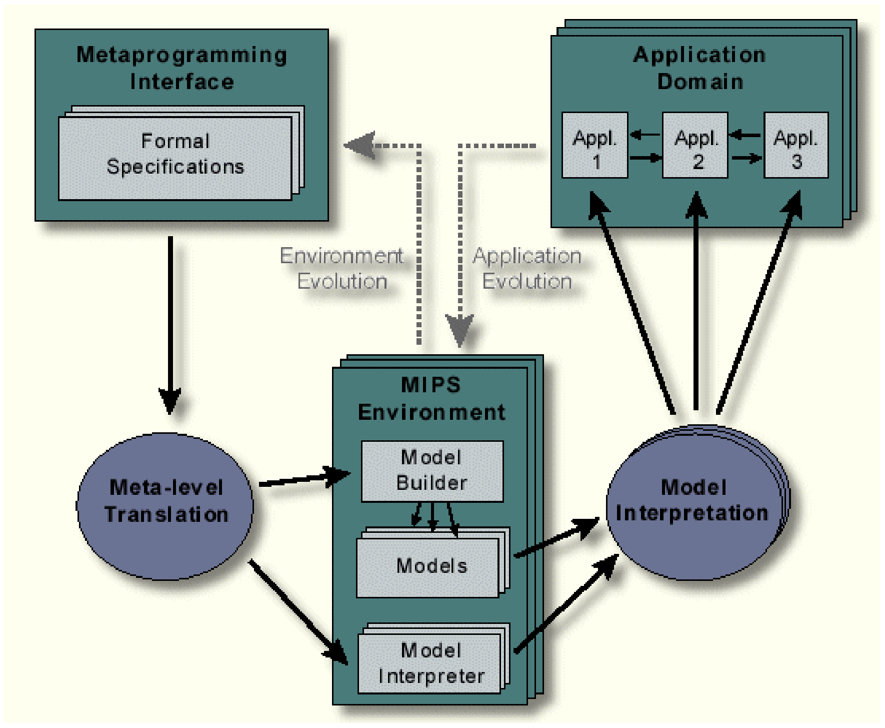


Figure 10 MIC – Applications Synthesis through the process of modeling [26]

The user builds a meta-model of the target domain for which the applications need to be modeled. The user can specify the set of entities that can be created in the target domain environment, their correct organization and interactions with other entities can also be specified in the meta-model. In other words, the meta-model specifies a modeling paradigm/language in terms of the syntactic, semantic and presentation information of the target application domains. Meta-level translation stage is responsible for interpretation of the meta-model and generating a configuration file for GME that is used to configure GME for the target environment. The model interpretation stage involves synthesis of applications from the user created models. The models generated using GME take the form of graphical, multi-aspect, attributed entity-relationship diagrams. The dynamic semantics of a model can be assigned during the model interpretation process. Using GME, large-scale, complex models that have hierarchy, multiple aspects, sets, references, and explicit constraints can be expressed.

MATLAB/Simulink

MATLAB, developed by Mathworks [70] is a high-level computing language that can be used for interactive algorithm development, data visualization, data analysis and numerical computations. MATLAB can be used in a wide range of applications some of which are financial modeling and analysis, signal and image processing, communications. It comes equipped with a range of toolboxes developed for specific domains like real-time systems, control systems, simulation among others. Along with MATLAB, Mathworks supports the Simulink product family. Simulink is a platform for multi-domain simulation. It provides an interactive graphical environment and a customizable set of block libraries that is extensible for specialized applications. MATLAB provides the ability to write a series of MATLAB/Simulink statements into a command file and then execute them with a single command. This feature provided by MATLAB/Simulink has been used towards the partial development of the work in the thesis.

CHAPTER III

DYNAMIC QoS MODELING ENVIRONMENT

This chapter presents the details of the design specification and semantics for the Dynamic QoS Modeling Environment (DQME) [79]. The chapter is broadly divided in two sections. The first section gives an overview of the DQME meta-model elements, their relationship with each other, their design specifications and their semantics. The second section presents details on the code generators that have been built for synthesizing code from the constructed domain models.

DQME Meta-Model

As discussed in Chapter 2, there are five essential questions that need to be addressed when developing a dynamically adaptive distributed system. The elements of the DQME paradigm address these questions for developing effective dynamic QoS adaptation.

"Mission" model is a DQME element that allows one to capture the QoS and functional goals of the system in a concise manner. Specification of what needs to be achieved by the system can be performed using the Mission models. Nonetheless, to be able to achieve the QoS goals, one needs to receive information regarding the current system state. The QoS "Observable" parameters provided in DQME facilitate in observing the current state of the system in QoS space. The QoS "Controllable" parameter element provided in DQME are "tunable" elements that cause the system to move from one point to the other in the QoS space. Depending on the current state of the system and the type of tunable parameters available, there may be multiple options for system adaptation. The interactions between these can be specified using the system dynamics. Among all the possible options available for system adaptation, the "Controller" models are responsible to select adaptation strategy. While making an adaptation decision, the controller takes into account the mission requirements of the system along with system dynamics.

A detailed explanation of the syntax and semantics of these meta-model elements is presented in the next few sections.

Mission Model

In the DRE systems, QoS objectives are dynamic in the sense that the notion of a preferred region of operation is linked with the notion of a mission or a mode for e.g., consider an aerial platform that participates in multiple missions, which may be concerned with surveillance, tracking, or weapons delivery. A distributed application performing information transfer with this aerial platform may have different QoS requirements for the information transfer, based on the nature of the mission.

A Mission Model in DQME captures specification of mission dependent QoS objectives in the form of preferred and mandatory bounds over values of QoS parameters. The mission model thus captures the high-level requirements of the system. Whenever there is a conflict in terms of satisfying multiple requirements, the mission model also allows the specification of relative importance over these QoS objectives.

Mission Model – Design Specification

These models express the high-level mission requirement as mentioned earlier. Figure 11 shows the meta-model of the mission modeling sub-language. The key concepts in this meta-model are *Missions<<Folder>>*, *Mission<<Model>>*, *Roles<<Model>>*, *Preference<<Connection>>* and *QPRef<<FCO>>*.

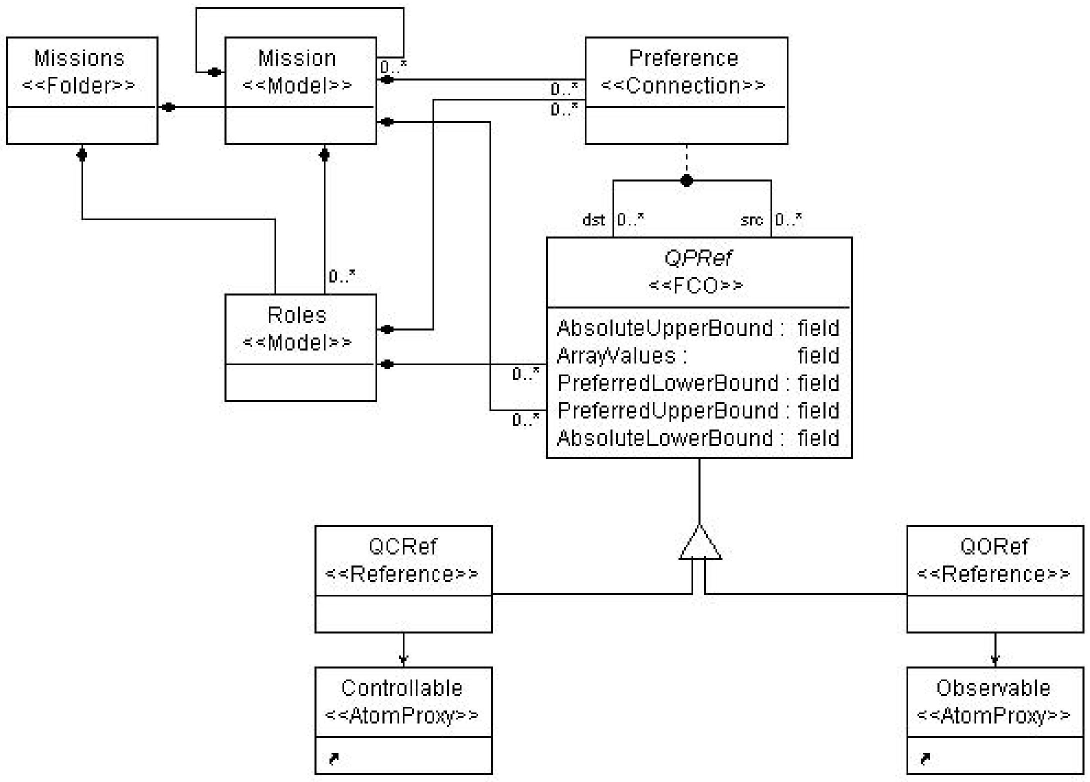


Figure 11 Mission model and QoS Parameters - Design Specification

Missions<<Folder>> (stereotyped as GME folder) are a collection of *Mission* objects, each of which encapsulates the mission specific QoS requirements. In a highly dynamic environment, a mission involves multiple participants each of which may have a different set of QoS requirements dependent on the role played by that participant (ex: a Reconnaissance UAV vs. a Combat UAV, both are UAV-s but play different roles). The

Roles<<Model>> allows specification of a set of roles each of which can then be assigned a set of QoS requirements. The QoS requirements are specified with a set of attributes defined over references to QoS parameters. The *AbsoluteLowerBound*, *AbsoluteUpperBound*, *PreferredLowerBound* and *PreferredUpperBound* attributes on the *QPRef*, an abstract base class concretized as QCRef and QORef which are references to QoS observable and QoS controllable parameters. The *Preference<<Connection>>* is an association class that allows expressing priorities in terms of the order of satisfaction of QoS requirements. References (a GME syntactic element) are used to specify QoS parameter to define the QoS requirements, since the QoS parameters are instantiated in the Application interface model (explained later in this chapter).

QoS Parameters

Observables and Controllable parameters (QoS Parameters) define the specification of the QoS parameters of interest for the system. Together, these constitute the QoS space of the system, and at any given point in the lifetime of the system, the value of the QoS parameters define the operating state of the system. Therefore, from a QoS perspective, one could view these parameters as state-variables. DQME partitions the set of QoS parameters into observables and controllables. It is important to note that the partition set of Controllable and the Observable QoS parameters is not disjoint. Some QoS parameters could be members of both these sets.

Observable Parameters

These constitute QoS parameters that can be observed or computed, through some service or provision in the underlying functional system. QoS parameters such as latency, bandwidth, CPU reservation, image quality, etc. are some examples of observable QoS parameters for some class of systems.

Controllable Parameters

These are QoS parameters that could be directly manipulated through some service provided by the underlying functional system. In an image processing system, image size, Image resolution, Signal detection threshold, are examples of QoS parameters that could be manipulated to accomplish some QoS objective. Latency is not a controllable parameter as it could not be directly manipulated, rather changing image size as a QoS parameter, may influence the latency of an image transmission application. Thus, controllable QoS parameters are the knobs available to the application for QoS adaptation. The Controllable QoS parameters in addition to being direct variables could also be in the form of packaged adaptations, such as those provided by QuO's Qosket encapsulation capability.

System Dynamics

The System Dynamics Models in DQME specify the potential trajectories of the system within the multi-dimensional QoS space. These specifications capture the dependency of the observable QoS parameters over the controllable and other observable QoS parameters, in the form of mathematical relations.

System Dynamic Model – Design Specification

The key concepts as shown in Figure 12 are *SystemDynamics<<Model>>*, *Function<<Model>>*, *DataVariable<<Atom>>*, *DataFlowConn<<Connection>>*. Figure 10 shows the meta-model of the System Dynamics.

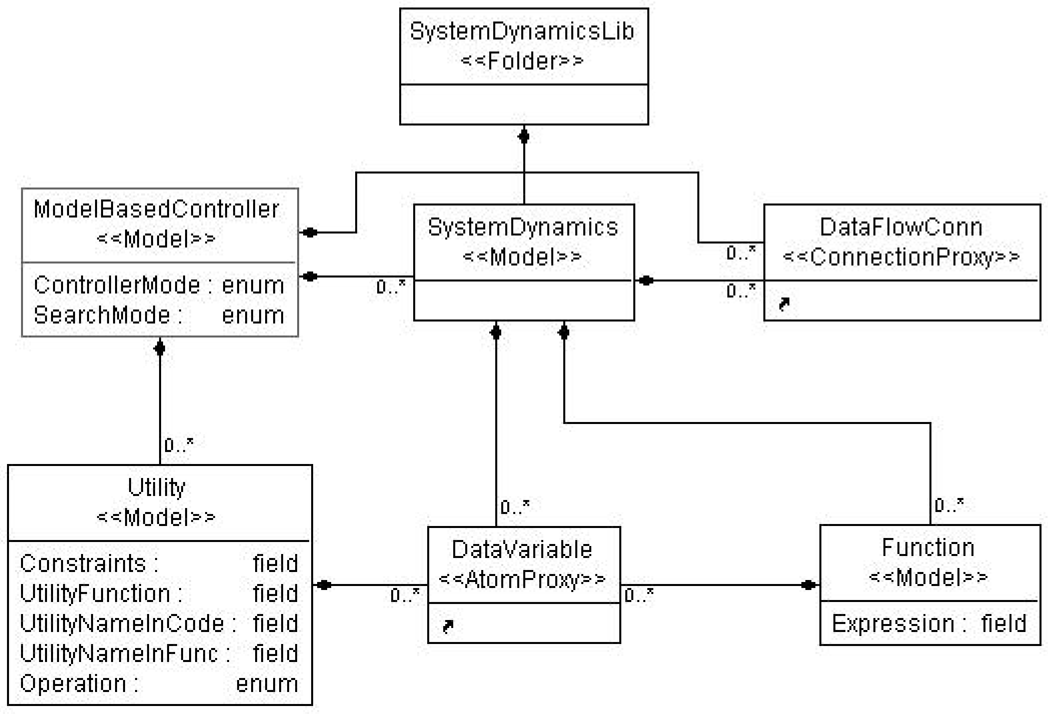


Figure 12 System Dynamics - Design Specification

The expression of the possible trajectories taken by the system is done using difference and differential equations expressed over a set of *DataVariables*, specifying the evolution of these variables over time. A general form of Function notation, without linking these directly to QoS Parameters has been used. The binding to the QoS Parameters is made externally, when the *SystemDynamics* is instantiated, by making associations between *QoSParameters* and *DataVariables* that form the port of the *SystemDynamics*. The *Expression* attribute on the *Function* model captures the mathematical relation.

System Adaptation

The system adaptation specification is the most important aspect of the QoS adaptation. These specifications represent the configuration of a parameterized controller from a suite of controllers available in the DQME. Primarily, DQME allows the use of a model-based controller to specify the adaptation.

The Model Based Controller (MBC) is primarily based on the OLC approach discussed in Chapter 2. It is configured by identifying the QoS space, the QoS objectives, the system dynamics, and additionally a utility function that characterizes each point in the QoS space with a utility value (or cost). The MBC utilizes these specifications to adapt and guide the system during operation in a trajectory such that the utility at any given point of time is maximized under the constraints posed by mandatory region specifications.

A discrete state-based controller is configured by defining a set of operating states, the conditions for transitioning from these states, and side-effects of taking the transitions. A Composite Controller (CC) is configured by expressing data propagation over a collection of controllers. Often, these take the form of set-point specifications i.e. an outer-loop (global) controller defines set-points for an inner-loop (local) controller.

System Adaptation Controllers – Design Specification

The Controllers design is specified using the Controller models. DQME allows representation of multiple control techniques. These include: Model-based Control, Discrete State-based Control, and a Composite Control that allows expression of hierarchical controllers, where individual controllers could be expressed with either of the above control techniques.

Figure 13 shows the Controller meta-model in DQME. The key concepts as can be seen in this meta-model are *State<<Model>>*, *CompositeController<<Model>>*, *ModelBasedController<<Model>>*, *DataVariable<<FCO>>*, *IOVar<<Atom>>*, *LocalVar<<Atom>>*, and *Transition<<Connection>>*.

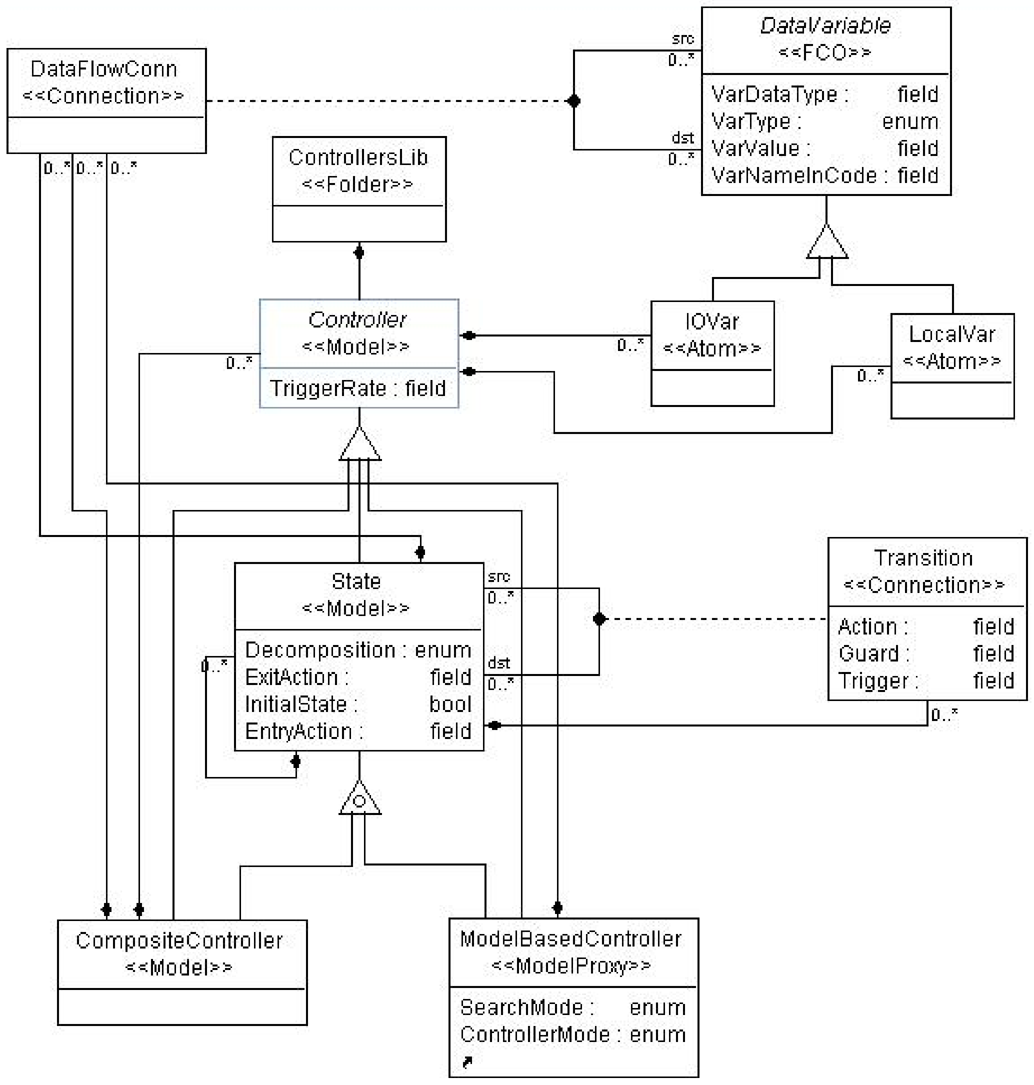


Figure 13 System Adaptation - Controllers design specifications

A specific type of inheritance known as interface inheritance between State, ModelBasedController (MBC), and CompositeController (CC) is used. The Interface Inheritance indicates that the derived class has the same interface as the base class and can play the role of the base class, however does not inherit the containment relations where the base class participates as a container. The implication here is that both MBC and CC can play the role of a State in defining a State Machine. The semantic implication is that the control action will be computed by the controller represented by the active state. Further details of the MBC, CC, State-based Controller, and Data and I/O variables follow.

Model Based Controller

The MBC controller is derived from the abstract Controller and interface inherited from the State (Figure 13). By virtue of inheritance from the abstract Controller, an MBC can contain *DataVariables*, which constitute the ports of the MBC to which QoS Parameters could be bound. An MBC contains *SystemDynamics* model and a *Utility* model (Figure 12).

A *Utility* model captures the utility function which assigns to each point in the QoS space a value. Variable passing between the *SystemDynamics*, *Utility*, and *DataVariables* is accomplished through the *DataFlowConn* connection. The MBC has an attribute "SearchMode" using which one can specify the possible type of searches in the QoS space. The two possible values that it can take are the "Relative Search" and the "Complete Search". Using the "Complete Search" option, searching in the entire space of possible values for QoS parameters is enabled. The search would return results that best satisfy the utility. In general, this type of search strategy can be used when there are bounded and discrete values for QoS parameters. However, the length and the resulting timeconsumption of this search strategy render the approach infeasible on account of the real-time constraints on the operation of the system. The "Relative Search" option is more practical. This search starts with some current values and searching is done in discrete steps from these values. The QoS Parameter values are modified to check how they can affect the utility before deciding on some particular values.

Discrete State-Based Controller

Often, a MBC is not adequate by itself to perform control in a highly discretized QoS space. DQME therefore allows expressing discrete state-based controllers. The representation is a hierarchical, concurrent state-machine based formalism, similar to Harel's Statecharts [69], as shown in the meta-model of Figure 13.

The key concepts from this perspective are *State<<Model>>*, and *Transition<<Connection>>*. The *Decomposition* attribute of the State indicates whether the contained states are composed to encapsulate a sequential behavior or concurrency. The attribute *isInitial* can be set to true for states which are the default states. The *EntryAction* and *ExitAction* attributes capture the effects of entering and leaving the particular state. The *Trigger, Guard,* and *Action* attributes on the *Transition* define the conditions for the transition to be enabled, and its effects.

Composite Controller

A Composite Controller allows composing multiple controllers, where the compositional semantics are those of dataflow. This type of control becomes relevant in expressing a collaboration of controllers in a hierarchical, inner-loop/outer-loop, or local/global type of configuration to solve an overall control problem. The key relevant modeling concepts are *CompositeController<<Model>>*, and *DataFlowConn<<Connection>>*. The containment relation between *CompositeController* and the abstract *Controller* class indicates that the *CompositeController* can contain any of the three controller types supported in DQME.

Data Variables

The DQME concept of *DataVariable* is sufficient for expressing dataflow interactions between controllers, functions, system dynamics, and utility. However, for reasons of preciseness in terms of scoping, and managing visual complexity, *DataVariable* has been expressed as an abstract concept, and specialized into *IOVar* and *LocalVar*. *IOVar* represents I/O variables of the container object, while *LocalVar* represents variables local to the scope of the container. While it is not apparent from the meta-model shown in Figure 13, the *IOVar* are defined to be visible as port of the container, while *LocalVar* are not visible as ports. This helps managing the visual clutter by avoiding several unnecessary ports, and also helps in preventing the user from making the mistake of making bindings to *LocalVar.*

Application Interface Modeling

The modularization of the QoS adaptation and a systematic integration with functional application design can be obtained using the Application interface modeling. This model allows encapsulating the functional aspects of the system and its interfacing to the QoS parameters. The key elements of this sub-language are described below.

Application Interface Model – Design Specification

Application Container

An application container (designated *AppContainer* in the meta-model) is an encapsulation of the functional application that serves as a wrapper facilitating the integration of the application with the QoS adaptation specification.

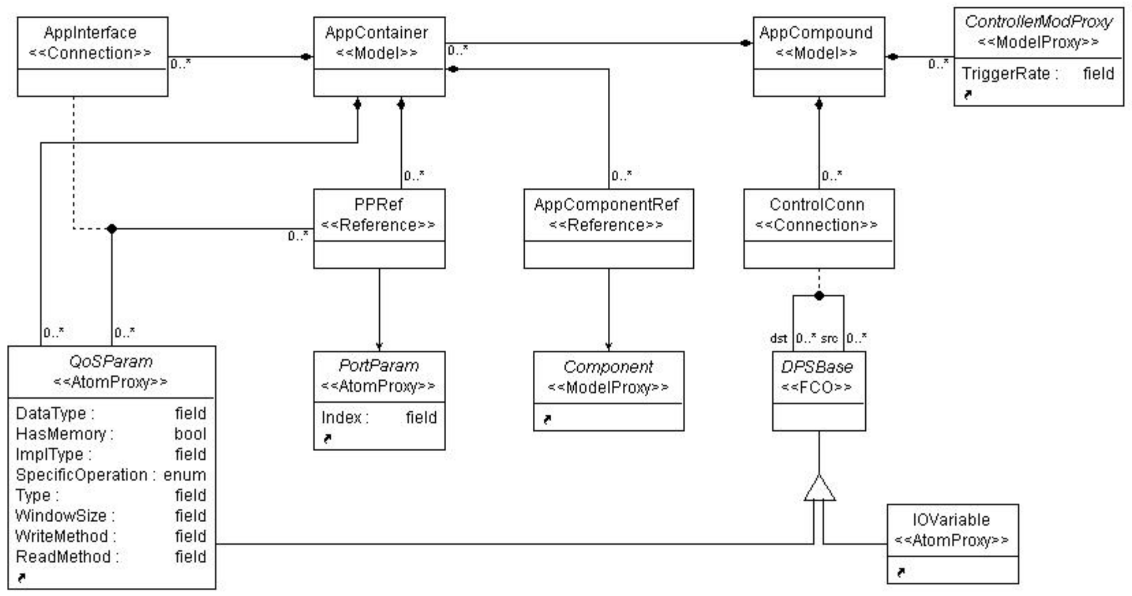


Figure 14 Application Interface Modeling

As seen in Figure 14, this container can contain the following objects:

- A reference to the application (*AppComponentRef*) that it is encapsulating. The use of Reference is motivated by the fact that the application model is defined elsewhere.

- References to ports and parameters (*PPRefs*) of the application, which are the interfaces through which the QoS adaptation can interact with the application. These interfaces are specific to the underlying functional design paradigm. A Software Radio Modeling Language (SRML) has been plugged into the existing version of DQME that has ports and parameters that can be used for the manipulation of the QoS information.
- Instances of QoS (Controllable and Observable) parameters. These are connected with the *PPRefs* to express the binding of the QoS Parameters to interfaces provided by the application. The *QoSParams* is stereotyped as an *AtomProxy* which refers to the actual *QoSParams<<Atom>>* defined in a different sub-language in the DQME meta-model.
- *AppInterface<<Connection>>* is the association between the *PPRefs* and the *QoSParams* as indicated above

Application Compound

An application compound (designated *AppCompound* in the meta-model) represents the composition of the QoS managed system. This container contains the controllers i.e. State-based, Model-based, or Composite Controller, an Application container (*AppContainer*), and *ControlConn<<Connection>>*. The *ControlConn* represents the flow of QoS information between the Controllers and the Application.

Translators Developed

Model creation needs to be followed by code generation for analysis, simulation and system assembly. Generated code provides a consistency that may be missing in code developed by multiple programmers on a team. Furthermore, the potential of introducing bugs in automatically generated code is much lesser as compared to hand-written code. Hand-coded objects pose a significant amount of risk for systems needing a high degree of security, persistence and extensibility. A bug in the generated code can be fixed by making appropriate changes to code generator that ensures the propagation of the fix throughout the system. If the generated code seems incorrect / insufficient in terms of the assigned semantics, the models can be iterated till a correct version of the code generator / generated code is obtained.

GME provides the users with the facility of interpreting the models created in a particular modeling environment and attach execution semantics to the models. It provides a framework through which the model user can extract information from the models in a desirable way. Thus, these models are used for synthesizing applications. GME provides various interpreter frameworks such as the Builder Object Network (BON), BON2, the Raw COM and the Visual Basic interfaces. Choice of various interfaces provides the user the flexibility to choose from the area of language expertise as implementation frameworks for the model translators.

BON2 based translators, for the automatic generation of MATLAB [70] (.m file) and C++ code, have been developed for the DQME meta-model so as to assign semantics for the domain models. Description of the translators developed in terms of their objectives and structural properties is presented below.

GME to C++ Code Generator

The C++ code generator elaborated here is responsible to generate C++ classes corresponding to the controller models created by the target-user of the system. The generated C++ code emulates the controller logic that can directly be used in the low level implementation frameworks where dynamic system adaptations need to be performed.

Structure of the C++ Code Generator

The interpreter does automatic C++ code generation for the controllers / resource managers modeled by the user for system adaptation. The generic strategy adopted for generation of C++ classes is by generating a C++ class corresponding to a controller designed by the user. This statement holds true for the Model Based Controller and the Composite Controller. However, the strategy adopted for generating a C++ class corresponding to a discrete State based controller differs from the above. The differences have been discussed in this section. All the classes, corresponding to the controllers, have an ‘init’ and an ‘exec’ functions for differentiating between the initialization and the execution semantics of the system.

Discrete State-based controllers allow the creation of other state-based controllers within themselves as they are based on the state-flow semantics. Therefore, generation of state-based controllers’ code needs to be done cautiously to ensure that the details regarding the active states, transitions and actions of the states are captured effectively. In the code generator provided for DQME, for every high level discrete State based controller, a C++ class is generated. But C++ code for all the other State based controllers that are contained within this parent is embedded within the parent State based class code. For effective representation of the Finite State machine semantics, the states within the parent state are represented as “enums”. A State variable controls the switching of control from one state to another. A switch-case statement is used to represent the details for different states. The transitions from one state to another have been represented as “if-else” conditions to set the state variable to a particular state. Therefore, when a transition is enabled, the “if” would evaluate to a true and the state variable would be set to the new state causing the control to move to the new “case”. The “guard”, if specified, will also be a part of the “if” statement generated so that the transition can only be taken when both the “transition” and the “guard” are true. The “onEntry” and the “onExit” conditions get represented in terms of functions. Unique functions are generated for every “onEntry” and “onExit” conditions for the states. The “action” attribute specifies the action to be taken when a particular transition is taken. Therefore, it has also been represented as functions in the C++ code. Names of functions generated are unique to every state for easy identification. These functions are invoked after the “onEntry” of that state.

The concurrent execution of the state machines has been represented in a pseudo-concurrent way. Depending on the number of Finite State Machines that would be present, those many number of state variables are generated for keeping track of the current active states in all the FSMs. The user/modeler bears the responsibility to name the data variables within the parent state in a unique fashion as all the data variables get represented in the same C++ class file duplication of which may lead to compilation errors. However, the user has the flexibility to reuse the data variable names within the Composite or the Model Based Controller as separate classes are generated for these controllers.

As the controllers can be modeled to represent data passing between them in terms of set-point specifications or just local data passing, appropriate getters and setters have been provided for the data variables in the public interface of the classes for enabling communication. A Model Based Controller or a Composite Controller present within a state is represented in terms of an instance of that particular class.

For every Composite Controller, an independent C++ class is generated. Since the composite controller is mainly used for depicting the dataflow semantics, functions are generated to depict the data flow. The order of dataflow between models is also considered when generating code for composite controllers.

Similarly, for every Model Based Controller, an equivalent C++ class is generated. The generation of code for the Model Based Controller is more complex than that for the other two controller types. The QoS adaptation mechanism is specified in this type of controller. The System Dynamics and the Function objects modeled in this controller are mainly responsible to depict the dynamics of the system in terms of mathematical expressions. The Utility model shows how the system will be driven towards the adaptation. As apparent from the meta-models above, the System Dynamics can consist of a number of function objects. Therefore, the code generated for the system dynamics objects represents the invocation of the functions generated (corresponding to the objects). The generated code has to take into consideration the order in which data flows between various function objects. If Function object A has been modeled such that the output variables from A are needed as inputs in B and C, then A has to be executed first in the generated code than B and C. Similarly, if Y receives inputs through its ports from W and X, then the code generated should ensure that invocation of W and X takes place before function Y is invoked in the code.

The function objects specified in the system dynamics are implemented as functions. The body of these functions is equivalent to what the user expresses in the "Expression" attribute of the object. The generation of code for the Utility objects is more complex. The evaluation of constraints should take place before anything else for the

Utility object is executed. The constraints can be specified using the “constraint” attribute on the Utility object. Thus the function generated corresponding to the evaluation would return a Boolean value. If the constraints are satisfied, then the Utility object specified using “UtilityFunction” is evaluated. Depending on the search strategy i.e. either a Relative Search or a Complete Search (can be specified for the MBC) is performed where the utility will be maximized or minimized as desired. Sample output of generated C++ code is shown in the next chapter.

GME to MATLAB Code Generator

Collective behavior of components can have dynamics that affect the system in some manner. Understanding the behavior of the system when it undergoes different mutations proves to be significant. Simulation of models can help us gain a better understanding of the working of the system under different resource constraints and environmental conditions. The translator presented here generates a MATLAB command file using which such simulations can be performed for the models developed using DQME.

Structure of the MATLAB Code Generator

This interpreter is also responsible for automatic code generation for the controllers / resource managers modeled by the user for system adaptation. However, this interpreter is geared towards the construction of a MATLAB command file rather than C++ classes. The command file generated contains a list of MATLAB Stateflow commands that when executed, produces the required MATLAB blocks. The actual logic for the generation of the blocks corresponding to every object found in the user-created model is implemented in parameterized helper scripts. The helper scripts are analogous to the functions serving certain goal. Therefore, the generation of the MATLAB file is reduced to the task of generating calls to the various helper scripts by passing appropriate parameters.

For every controller object in the user model, an equivalent call to the "CreateControllerModel" helper script is generated. This helper script needs multiple parameters such as the name of the controller, state chart information for this controller and positioning co-ordinates. Since, the controller and all the objects within the controller are represented as states, a statechart is created for every controller. This statechart further has various finite state machines (FSM) and/or concurrent FSMs. The Statechart information that needs to be generated consists of the input and output variables list, input and output variables initial values list and the input and output ports list, states list, parents of the states list (i.e. a list of states containing these states), a list of transitions and a list of states that contain these transitions.

The interpreter assigns Stateflow semantics to all the controllers. Consequently, the objects are converted to appropriate hierarchical States. For representing concurrent execution of FSMs, dummy parent states are created. The creation of dummy parent state enables the setting of the type of this parent to "PARALLEL_AND" so that the internal states could concurrently run. For the other states that do not need parallel execution, their types are set to "EXCLUSIVE_OR". For the MBC, separate files are created based on the strategy explained above where the evaluation of the constraints is done prior to evaluation and search strategies specify the type of search in the QoS space.

The "CreateConnection" is also a parameterized helper script that is responsible to create connections between the ports of the controllers and the application container. The

"CreatePlantModel" script generates blocks for application containers. All of the helper scripts apart from the "CreateConnection" script needs positioning information as one of their parameters. Generation of correct positioning information is required to ensure that the states are correctly contained within their parents as MATLAB relies on the borders of the states to determine the child-parent state relationship.

For every Model Based Controller, a separate command (.m) MATLAB file is generated. The approach taken to generate code for the Model Based Controller is similar to the approach adopted for generating C++ code for MBCs. The generated code has to take into consideration the order in which data flows between various function objects. If Function object A has been modeled such that the output variables from A are needed as inputs in B and C, then A has to be executed first in the generated code than B and C.

Similar to C++ code generation, the function objects specified in the system dynamics are represented as functions in the .m file. The body of these functions is equivalent to what the user expresses in the "Expression" attribute of the object. The evaluation of constraints takes place before anything else for the Utility object is executed. If the constraints are satisfied, then the Utility object specified using "UtilityFunction" is evaluated. Depending on the search strategy i.e. either a Relative Search or a Complete Search (can be specified for the MBC) is performed where the utility will be maximized or minimized as desired. Sample output of generated Simulink code is shown in the next chapter.

Overall Scenario Usage of DQME

Figure 15 shows where DQME fits in the overall scenario. The user is required to create design time models, by constructing the mission models, specifying the tradeoffs and the QoS adaptation mechanisms. The code generators are responsible to develop code that can be used in appropriate environments. At the meta-level translation stage, the meta-interpreter tool is used to interpret the meta-models and generate a target configuration file for GME. The model weaver is a tool developed at ISIS, Vanderbilt University that accepts as inputs an xml file that is an export of the models created and a specification file provided by the modeler and can weave out a new or enhanced xml file containing information pertaining to the enhanced models and that meets the specifications given in the file (More information about Model Weaver is at [82]).

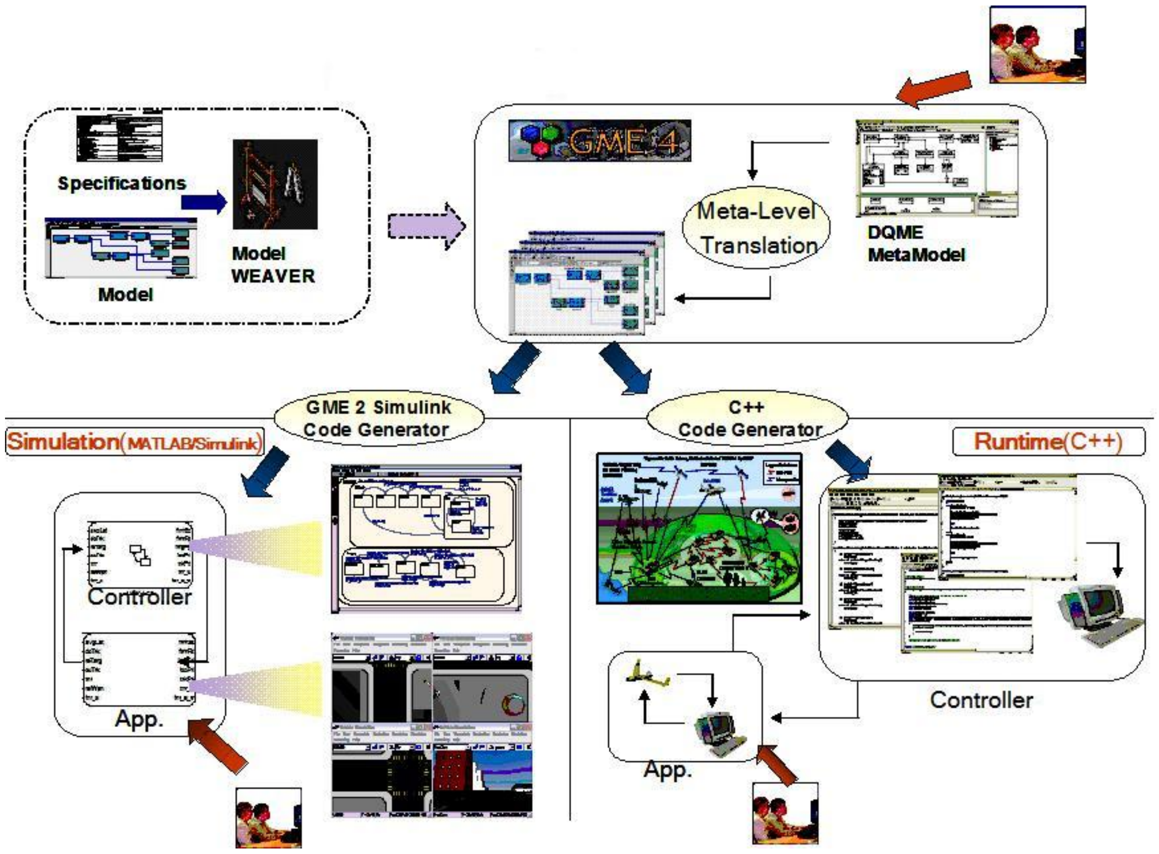


Figure 15 Overall Scenario Usage for DQME

CHAPTER IV

CASE STUDY – PCES CAPSTONE DEMO

This chapter presents a case study of using the DQME paradigm to model the QoS requirements of a mission-critical, real-time system. The DARPA PCES Program's capstone demonstration was used to show the efficacy of designing a QoS-specified multi-UAV surveillance and target-tracking applications with real-time requirements. The following section presents the study-scenario and illustrates the real-time requirements of the time-critical targets.

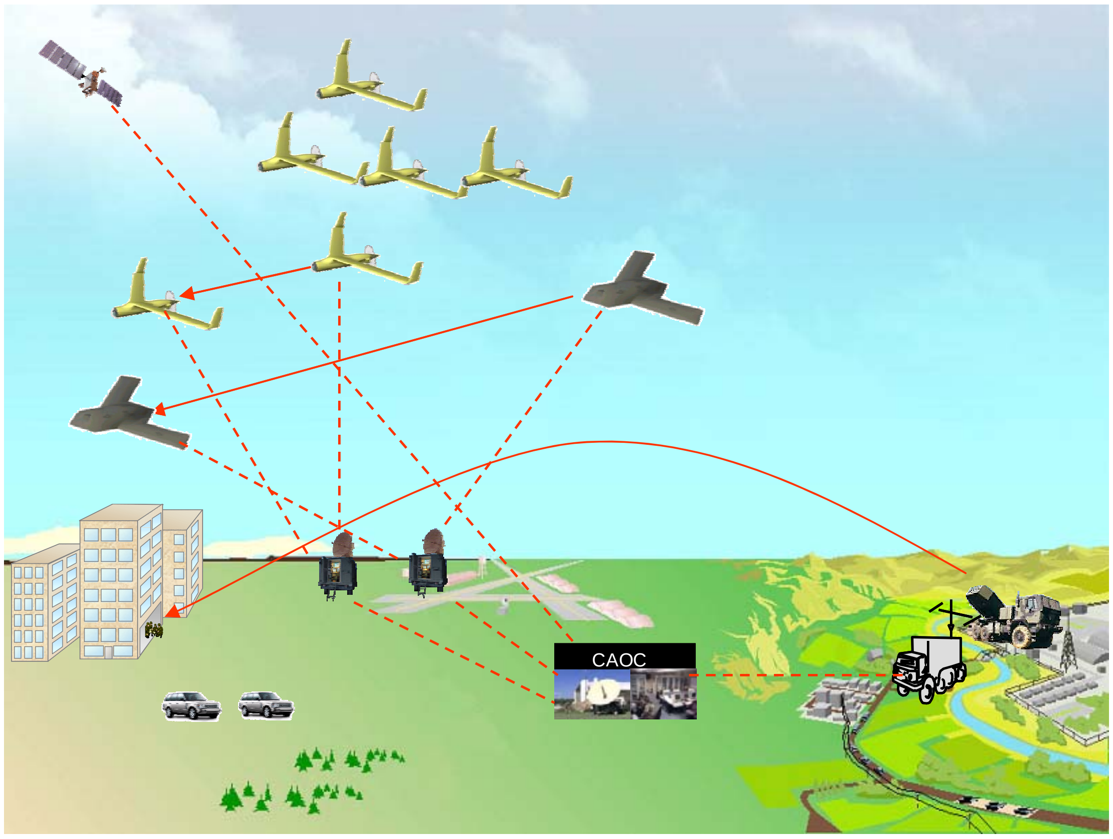


Figure 16 The PCES Capstone demonstration scenario

Scenario Description

The PCES Capstone demonstration scenario consists of a set of Reconnaissance Unmanned Aerial Vehicles (RUAVs) responsible for performing theater-wide surveillance and target tracking, a set of Combat UAVs (UCAVs) that act as weaponized UAVs and ground vehicles for time critical targeting. Each UAV has a camera mounted on it to capture the surrounding images. The surveillance imagery, captured by the camera mounted on the UAVs, is sent to a Command and Control (C2) Center or a Combined Air Operations Center (CAOC). This imagery is then analyzed and commands transmitted to the RUAVs. RUAVs can be commanded to focus on certain areas of interest (AOI). When a commander finds a certain AOI such as a threat or some target, he can command the RUAV to concentrate surveillance at that AOI. On a positive identification of a threat, the commander dispatches a ground or an air combat unit to engage the target, with the UCAV performing battle damage assessment afterwards.

A system comprising various components that need to interact with each other in order to achieve certain goals can be identified in this scenario. For example, the image sent by the RUAV that finally gets displayed to the commander's display at the CAOC travels through a set of distributed nodes before reaching the display. These nodes may be running on different platforms with different operating systems and may communicate using various network protocols, thus constituting a heterogeneous system. The nodes may be considered to form an end-to-end application string that must be traversed by the image to reach its destination – the target image sensor or the commander's display. In a distributed system depicted in the picture above, there are multiple such application strings corresponding to multiple UAVs. A significant challenge lies in the management of end-to-end QoS requirements of these application strings because of the dynamic and heterogeneous nature of the applications. The following sections discuss the details involved in modeling the end-to-end QoS adaptation strategies using DQME for this scenario.

Modeling of the PCES Capstone Demo

Brief Description

Key elements identified for QoS adaptation modeling in this situation involves the following:

Roles and Mission Requirements

UAV plays three primary roles in this scenario: Surveillance, Target Tracking and Battle Damage Assessment. Each of these roles has certain mission requirements associated with each of them:

- Surveillance: In this role, the RUAV performs the theater-wide surveillance. An important mission requirement is to maximize the surveillance area along with maintaining appropriate resolution of the imagery that is being sent to the display of the commander to make an easy identification of the AOI. For enabling an easy identification of threat, the speed at which the imagery is sent is also critical. The time lag between two images arriving at the display may thwart the purpose and prove detrimental to the mission. The maximum possible rate, image size, and resolution are determined by the capabilities of the camera on the RUAV. The adaptation across RUAVs will take into account the number of UAVs and the amount of resources. The tradeoffs also have been identified along with these mission requirements and have been discussed in the next subsection.

- Target Tracking: In this role, the RUAV has been identified to be observing an AOI and hence performs target tracking. Obvious importance need to be given to this RUAV (or set of RUAVs) over others. The mission requirements of RUAV that enters the Target Tracking role, is to provide high resolution imagery using which the human operator/commander can positively identify target or threat. Thus, in the case of the RUAV now hovering upon the AOI, the minimum rate is no longer dependent on the speed of the RUAV, but by the speed of any mobile targets (or more accurately, the difference between their speed and that of the target tracking RUAV). Likewise, if the targets are stationary and the target tracking RUAV is centered on the AOI, cropping the image to remove peripheral or less/un-important imagery using a different scan size can be an option.
- Battle Damage Indication: The UAV enters this role when a target is engaged to perform the Battle Damage Indication. The UAV needs to provide imagery consistently and continuously until a human operator can determine that there is sufficient detail to discern battle damage. High resolution imagery needs to be provided in this case.

Figure 17 shows the complete Mission Modeling done for Capstone in GME.

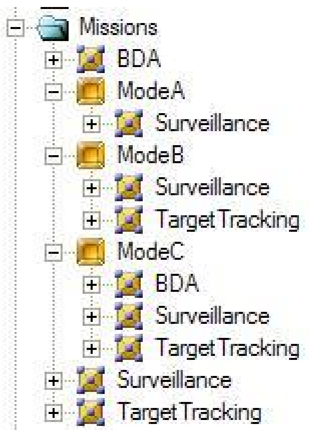


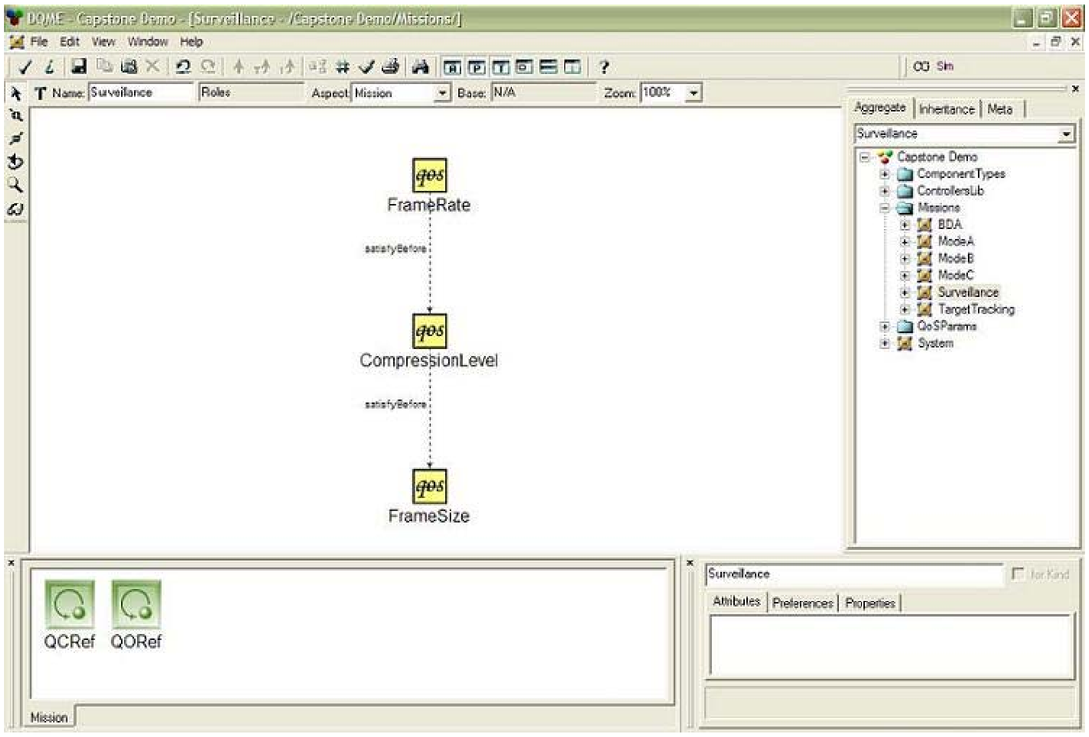
Figure 17 Mission Models for Capstone Demonstration

Additional modes, ModeA, ModeB and ModeC were specified to be able to model multiple UAVs in different roles at any instant of time.

Trade-Offs

Occasionally, in an attempt to satisfy a certain QoS goal the other parameter values may get disturbed. For example, to satisfy a QoS goal of sending compressed images, CPU resources utilization may increase. Under such constraining conditions, tradeoffs needs to be specified to determine which parameter values/goals should be given more importance than the others. The tradeoffs for the role descriptions of the UAVs are described below. The decisions of the tradeoffs were made by the sponsors after discussing with the domain experts.

- Surveillance: In this mode of operation, the precedence order of tradeoff is resolution followed by size and then the rate. The “satisfyBefore” label signifies the ordering. The label starts from the FrameRate towards compression level and then the FrameRate implies that a UAV in surveillance mode (RUAV) is least concerned about the rate at which the images are transferred whereas the resolution of the images is of paramount importance. In case a choice is required to be made for trade-offs, the application would compromise the rate of transmission rather than the image resolution. Image compression in general changes the overall data size and image processing time. Lossy compression can also affect (reduce) image resolution.


DQME - Capstone Demo - [Surveillance - /Capstone Demo/Missions/]

Name: Surveillance
Roles
Aspect: Mission
Base: N/A
Zoom: 100%

Figure 18 Surveillance - Trade-off Modeling

- Target Tracking: In this mode, assuming a stationary AOI, the tradeoff order for the privileged target tracking RUAV is the rate followed by the scan size i.e., changing the image size and resolution are not allowed. The minimum frame rate can be even lower than one frame every six seconds, but the image size and resolution cannot be lower than the minimums for surveillance role.

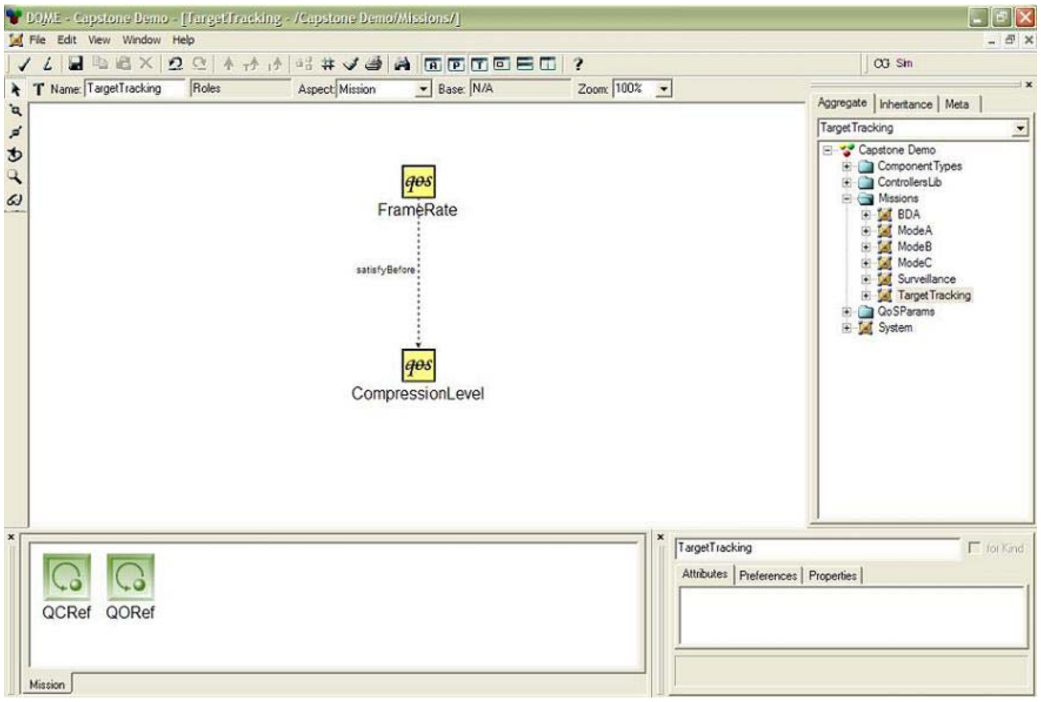


Figure 19 Target Tracking - Tradeoffs Modeling

- Battle Damage Indication: The tradeoff order for the RUAV in the battle damage assessment mode of the imagery is the frame rate followed by the image size. While doing the Battle Damage assessment, the rate at which the images are sent are more important than the size of the image sent.

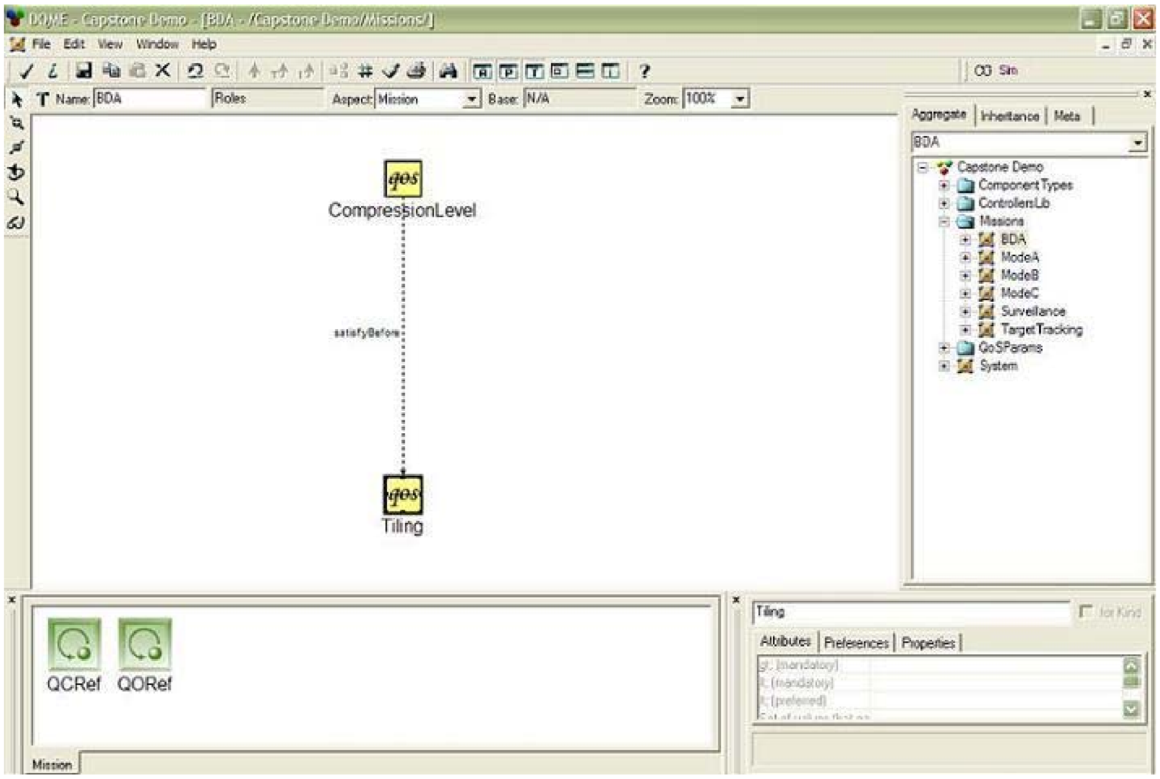


Figure 20 BDA - Trade-off Modeling

QoS Parameters

Depending on the roles performed by the UAVs and the mission requirements that need to be adhered to, following are the set of QoS parameters identified for this scenario.

<u>QoS – Observable Parameter Modeling</u>

Observable Parameters, in the context of the Capstone model, specify the metrics that can be measured at runtime to help calculate the current QoS. This part of the model also selects the appropriate behavior to invoke, and subsequently measures the effectiveness of the adaptation. These include:

- Mission attributes like the bandwidth allocated, roles of participants and allocated CPU

- Physical attributes of the UAV like the speed of the UAV, scan size, frame rate, image size, and resolution of the UAV camera
- Run time data attributes that include actual frame rate, image size, and resolution
- Attributes that correspond to the resources like the bandwidth capacity, bandwidth used and CPU reservation levels

QoS – Controllable Parameter Modeling

The Controllable Parameters specify the aspects of the model that can be modified at runtime to change the QoS. These are the "knobs" through which the changes can be done to the system. A number of controllable QoS mechanisms and other adaptive behaviors were identified and have been represented in the model. These include:

- CPU Reservation Level and CPU Priorities
- Image compression using various compression algorithms
- Image scaling
- Image cropping

Figure 21 gives a snapshot of the parameters modeled.

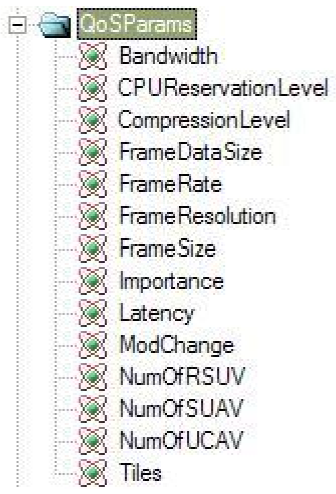


Figure 21 QoS Parameters Modeling

Resource Managers

The QoS management is based on a multi-layered structure. The important elements of the QoS management structure are:

- QoS adaptation mechanisms
- System level/Global Resource Manager
- Local Resource Managers

The Global resource manager (Figure 22) is responsible to assign bandwidth and CPU to each UAV. While computing these values for each UAV, the controller needs to takes into account the roles/importance of each UAV along with the amount of shared resources available in the system.

The Local Resource Managers accept the inputs (also referred to as set-point specifications) of the estimated values by the System Resource manager and determine new values depending on the local adaptation scheme available for effective performance of the UAV in that role. The global resource manager is modeled as a Composite Controller. Composite Controllers enable easy representation of compositional semantics and dataflow in the contained controllers. The local resource managers are modeled as Model Based Controller for representing the set of operating states, conditions for transitions, side-effects of these transitions and internally representing the System Dynamics and Utility models. The system level resource manager has been designed as an outer-loop controller that defines set-points for a local level resource manager designed as an inner-loop controller.

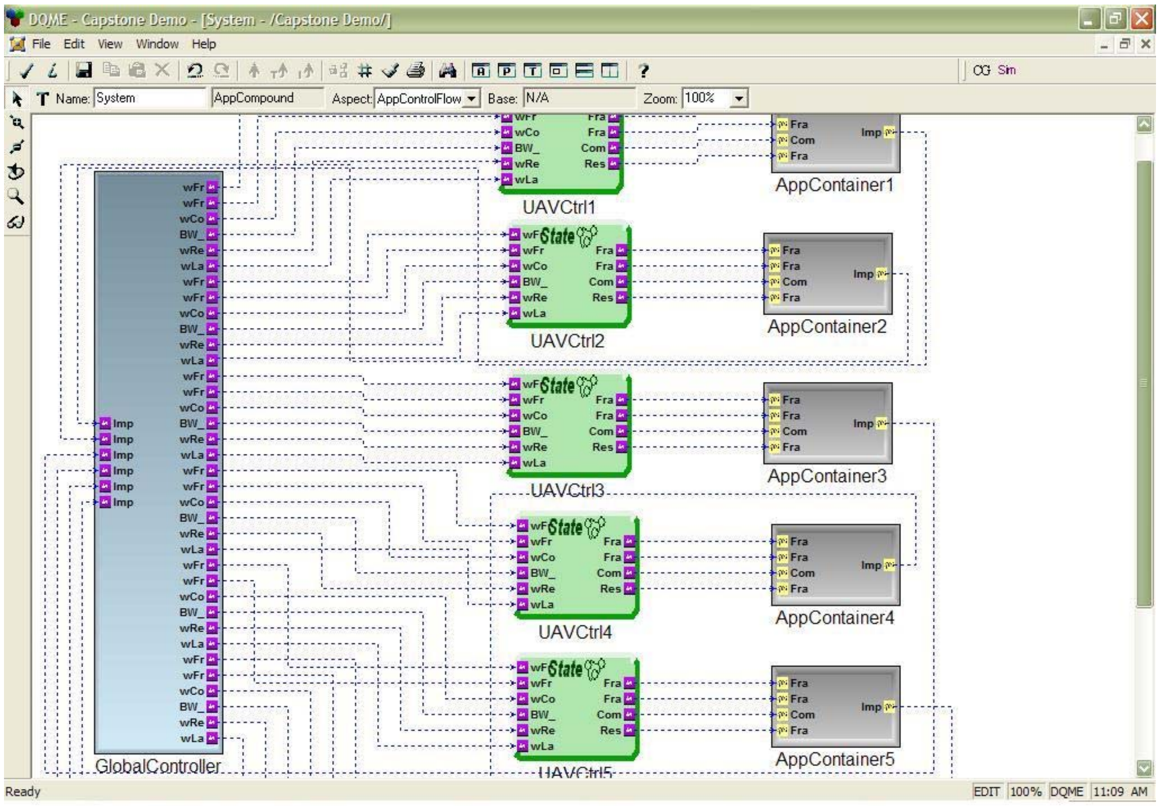


Figure 22 Capstone modeling of various resource managers

System Dynamics Modeling

System dynamics in the Capstone project are represented using the state-space representation. The QoS parameters, discussed above, represent the n-dimensional QoS space. As a consequence, the system dynamics model represents all potential trajectories that can be taken by the system in the multi-dimensional QoS space. The model has been encapsulated at two different levels and captures the relation between the number of UAVs, the available resources and the roles of the UAVs. This relation shows the effect on bandwidth availability and the CPU that could get allocated to various UAVs.

The local level dynamics consider situations where an increase of usage of a particular resource may lead to an increase/decrease of some other. For e.g., invoking compression operation on an image will reduce the bandwidth usage but will increase the CPU usage. Effects of this nature make capturing the local-level system dynamics more complicated. Modeling the system dynamics normally needs an experimental and analytical understanding of how particular adaptation strategies may affect the observable parameters which would in turn affect the UAVs. The interactions between these are captured and specified in the system dynamics model using mathematical relations. The System Dynamics have been captured using the "Function" models for modeling the system dynamics. The following picture depicts the usage of the "Function" objects to capture the system dynamics.

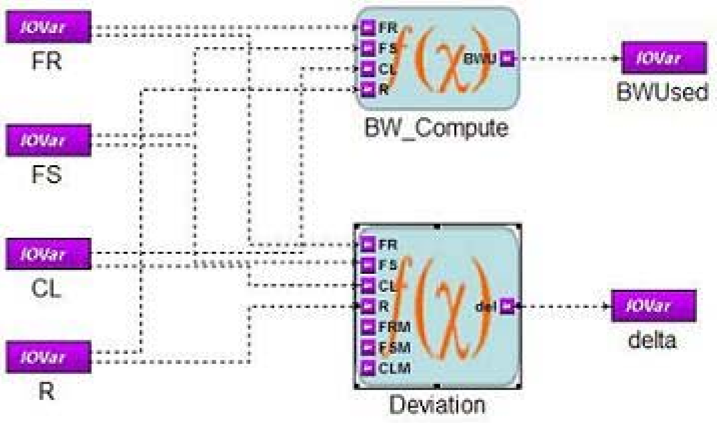
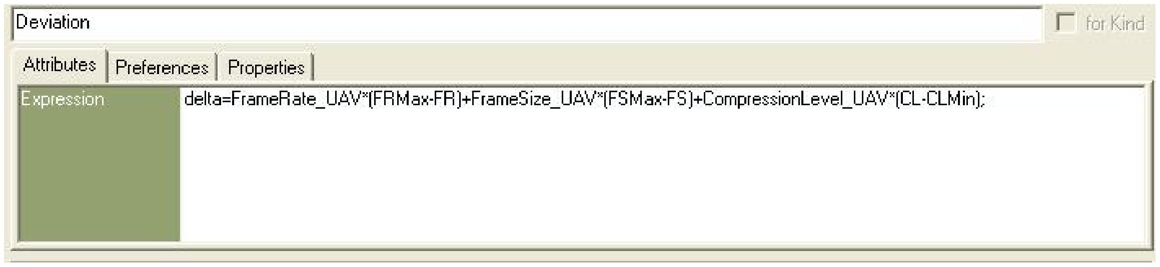


Figure 23 System Dynamics representation using the function models

The mathematical expression is captured using the “Expression” attribute present on the “Function” models. For the above represented Deviation function, the expression value is expressed as:

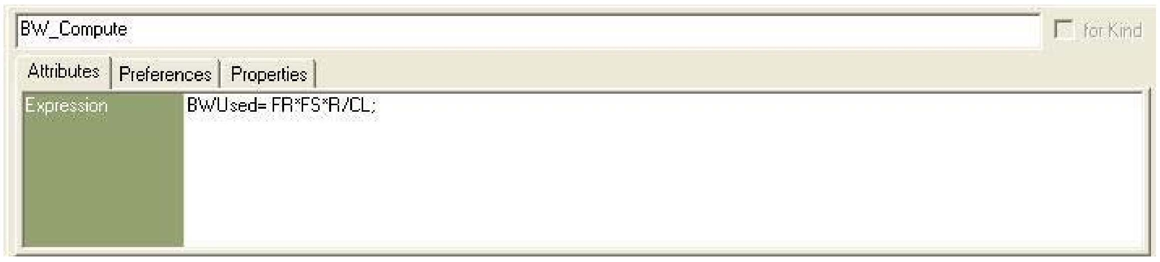

Deviation

Expression

```
delta=FrameRate_UAV*(FRMax-FR)+FrameSize_UAV*(FSMax-FS)+CompressionLevel_UAV*(CL-CLMin);
```

Figure 24 "Deviation" function expression

For the above represented BW_Compute function, the expression value is:


BW_Compute

Expression: BWUsed= FR*FS*R/CL;

Figure 25 "BW_Compute" function expression

Adaptation Strategies

The adaptation strategies specified in the local resource managers uses the frame rate, size, compression level, resolution and the allocated bandwidth (specified in terms of set-point specifications by the global resource manager).

The following figure depicts the usage of System Dynamics and the Utility objects for representation of the adaptation strategies in the local resource managers.

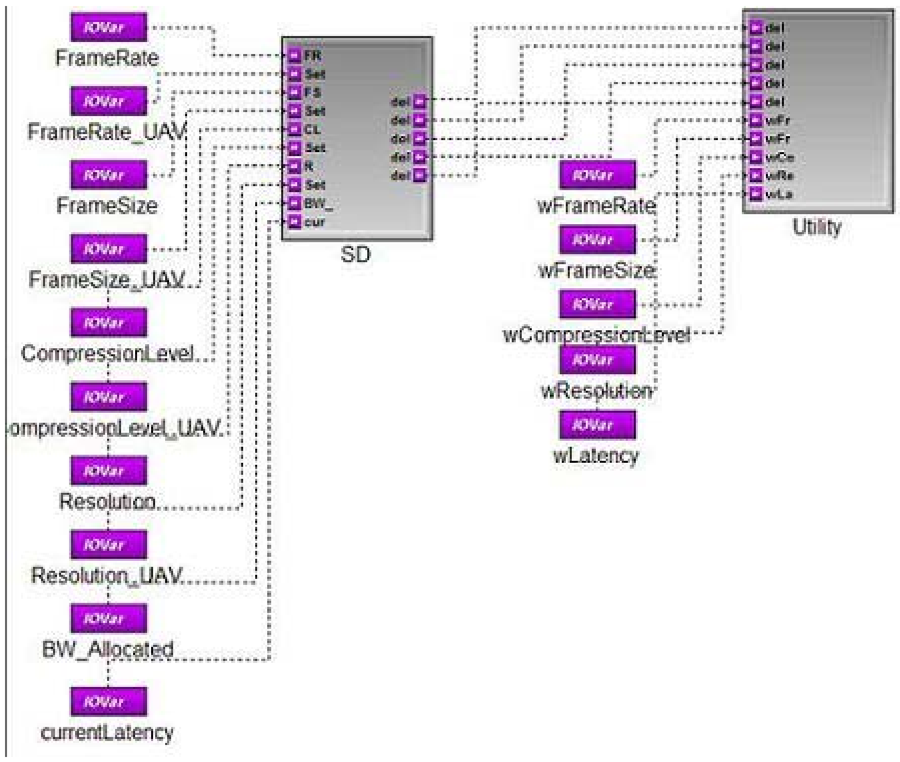
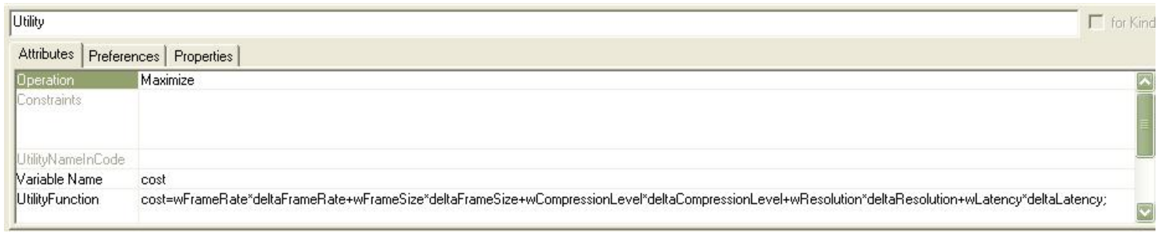


Figure 26 System Dynamics and Utility driven QoS adaptation representation

The "Utility" model shown in the picture above forms an important part of the QoS management and is responsible to drive the QoS adaptation in terms of achieving improved QoS. The following picture depicts the "Utility" model attributes captured:


Utility

Operation	Maximize
Constraints	
UtilityNameInCode	
Variable Name	cost
UtilityFunction	cost=wFrameRate*deltaFrameRate+wFrameSize*deltaFrameSize+wCompressionLevel*deltaCompressionLevel+wResolution*deltaResolution+wLatency*deltaLatency;

Figure 27 Utility model – Attributes

Code Synthesis from Models

Synthesis of code was performed by executing the code generators that were developed towards the development of DQME atop the constructed models. The sample outputs of the code generators are presented in this section.

C++ Code Generator – Sample Output

The output of this code generator is a set of classes representing the controllers designed by the user. An example model and the equivalent generated code have been shown below.

The model shows the "Role_Priority_Determination" model that has 6 concurrent FSMs contained. (Role_Priority_Determination_1, Role_Priority_Determination_2 - Role_Priority_Determination_6)

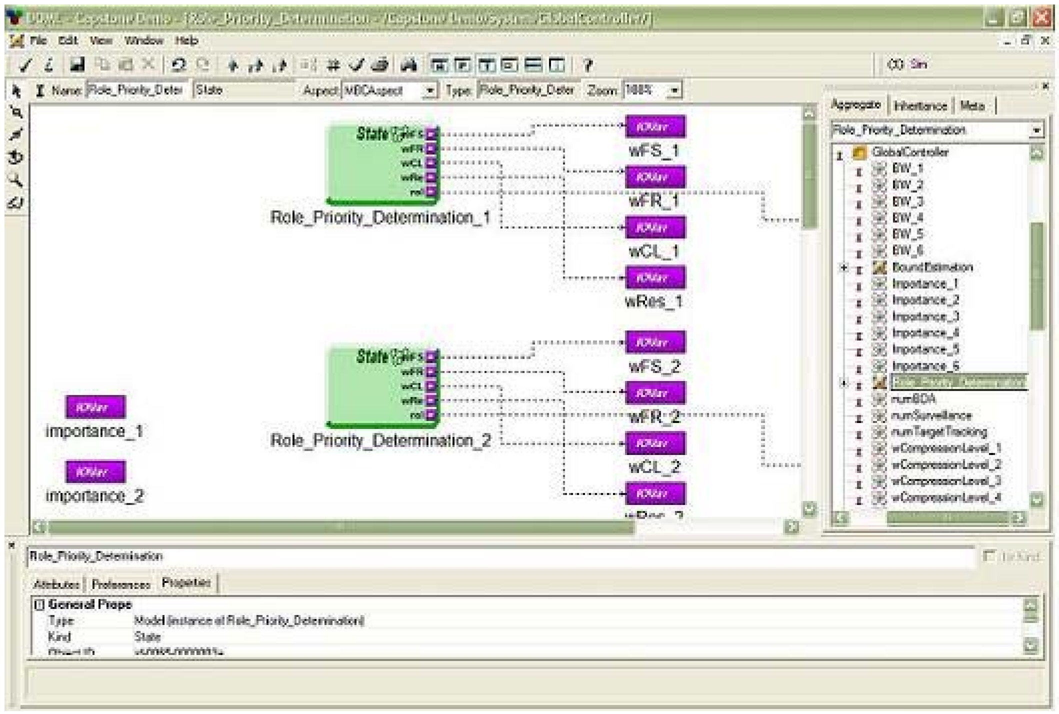


Figure 28 Example Model

The sample partial output of the code for this model object is seen in Table 1

Table 1. Sample C++ code to show the init(), exec() and pseudo-concurrent executions of FSMs

```
//Init function impl
void Role_Priority_Determination_GlobalController::init()
{
//Initialize the state vars if any
this->StateVar_1=InitialRest;
this->StateVar_2=Surveillance2;
this->StateVar_3=Surveillance1;
this->StateVar_4=Surveillance5;
this->StateVar_5=Surveillance4;
this->StateVar_6=Surveillance3;
}

//Exec function impl
void           Role_Priority_Determination_GlobalController::exec(int
importance_6,int  importance_5,int  importance_4,int  importance_3,int
importance_2,int  importance_1,int  numBDA,int  numTargetTracking,int
numSurveillance)
{
this->importance_6=importance_6;
this->importance_5=importance_5;
this->importance_4=importance_4;
…….
this->numTargetTracking=numTargetTracking;
this->numSurveillance=numSurveillance;
exec_dataflow1();
exec_concFSM_1();
……..
exec_concFSM_6();

//Assigning the output vars of this State Controller
this->role6=role6;
………
this->wFS_1=wFS;
this->wRes_6=wRes;

}
```

Table 2. Sample C++ code for FSM execution

```
//Extra func impls for conc FSMs execution

void Role_Priority_Determination_GlobalController::exec_concFSM_1()
{
switch(StateVar_1)
{
case BDA:
if((importance_6 <= 0.5))
{
BDA_exec_onExit();
BDA_InitialRest_transitionAction();
InitialRest_exec_onEnter();
StateVar_1=InitialRest;
}
else
{
BDA_exec_during();
}
break;
case InitialRest:
if((importance_6 > 0.5))
{
InitialRest_exec_onExit();
InitialRest_BDA_transitionAction();
BDA_exec_onEnter();
StateVar_1=BDA;
}
else
{
InitialRest_exec_during();
}
break;

}
```

MATLAB Code Generator – Sample Output

The output of the generator is a set of .m i.e. MATLAB files. These files contain commands for the creation of the states, function blocks, ports and the respective connections. Table 3 shows partial sample of the generated .m file corresponding to Role_Priority_Determination model shown above. In the sample output below, the state names starting with "DP_" are dummy states that are created so that their "type" could be set to “PARALLEL_AND” which would enable concurrency for the states contained within it.

Table 3. Sample Matlab code to show the object equivalent states and dummy states

```
%States Information for GlobalController

St_0=struct('name','GlobalController','position',[50,50,400,400],'ini
tial','0','labelen','','labeldu','','labelex','','decomposition','EXC
LUSIVE_OR','vars',{{'Importance_4','Importance_3','Importance_2','Imp
ortance_1','Importance_6'……..});

DP_St_0=struct('name','GlobalController_dp0','position',[50,50,100,10
0],'initial','1','labelen','Importance_4=GlobalController.Importance_
4;Importance_3=GlobalController.Importance_3;……..}});
.
.
.

St_8=struct('name','Role_Priority_Determination','position',[50,450,1
00,100],'initial','0','labelen','importance_4=GlobalController_dp0.Im
portance_4;

DP_St_1=struct('name','Role_Priority_Determination_dp1','position',[5
0,50,100,100],'initial','1','labelen','importance_4=Role_Priority_Det
ermination.importance_4; Determination.importance_3…..}};

St_27=struct('name','BDA','position',[100,100,200,175],'initial','0',
'labelen','','labeldu','','labelex','','decomposition','EXCLUSIVE_OR'
,'vars',{{}},'initialVarValue',{{}});
```

Case Study – Evaluation

The PCES Capstone demonstration case-study, which is an instance of real-time scenario of DRE systems, helps make apparent the advantages of using DQME to model QoS adaptive applications. Using DQME, a clear separation of QoS properties from the functional aspects is provided. Expression of the system dynamics is made easier with the help of the models. Using the Function models, the data propagation and interactions and the mathematical expression for the system was easily captured.

The overall complexity of the models can be measured by doing some quantitative analysis. 7 System Dynamics models are constructed to capture the system and local level dynamics in terms of the Utility models for the adaptation strategies specification. The system dynamics and the adaptation strategies specification at the system level were captured using over 120 data variables. For each of the local level system dynamics, an approximate of 100 data variables is required thus adding up to approximately 800 data variables in the entire model. The model on a whole is made up of over 500 data flow connections to enable transfer of data. Developing complex models as these is simplified using DQME.

The domain engineer bereft of control-theory knowledge finds it intuitive to express the QoS as it has been elevated to a higher abstraction level. Behavioral modeling along with the resource management strategies under varying operating conditions are easily captured at the early stages of design time.

Code generators were used on these models for synthesizing the appropriate controller logic. BBN's environment supports a set of generic components and runtime libraries that aid in run-time adaptation and control of a general class of distributed systems. Collectively, these pieces form a fairly robust system for QoS adaptation. The MATLAB code synthesized from the MATLAB code generator was successfully applied towards the simulations of the PCES Capstone demo in Simulink. More details can be found in the PCES demo presentations [83]. Covering the details of the MATLAB Simulation is beyond the scope of this thesis as the simulations were worked on by a different member in the team.

CHAPTER V

CONCLUSION AND FUTURE WORK

Conclusion

Design and development of large-scale DRE systems has been difficult primarily due to factors such as the real-time nature of the applications, distribution of the components over variety of nodes that may have resource constraints and unpredictable network and environmental conditions. Since all these factors contribute towards the design complexity of the DRE systems it is necessary to maintain a degree of QoS in these systems that will make certain guarantees about their performances.

As mentioned earlier, control engineers have sufficient control background but lack domain knowledge. On the other hand, domain engineers, who have significant amount of knowledge of the functional aspects of the system, do not have enough background of control-engineering to design and develop QoS adaptive applications. DQME, a modeling framework, developed and presented in this thesis provides a way for the domain engineers to be able to develop QoS adaptive applications.

The solution presented is a modeling framework that offers a graphical way of describing all the details of the systems at an abstract level and their relationships in a precise way. Modeling also enables the anticipation of consequences of multiple interactions. Additionally, unlike the low-level techniques used for developing systems that are cumbersome to manage when the systems increase in complexity, modeling offers not only a manageable solution but also reduce the development time.

As seen in the existing literature for building QoS in DRE systems, the adaptation strategies employed thus far, are largely ad-hoc. QuO project developed at BBN offers a separation of concerns between the functional and QoS goals of the system; however it still uses ad-hoc schemes for the adaptations. DQME offers a way to specify interactions between various QoS variables in a precise way.

DQME elevates the control-engineering aspect of building QoS in a system to a higher level of abstraction that is amenable and can be used by the domain engineers to develop QoS-adaptive applications without needing significant amount of controls knowledge. DQME has been specified at a meta-level to enable development of QoS in large class of DRE systems.

DQME focuses primarily on the QoS aspect of the system, and allows the users to plug in various meta-models representing different functional modeling and composition techniques. Thus, it is able to offer a separation of concerns between the QoS and functional goals of the systems. DQME supports modeling of various concepts required for building QoS adaptation in the systems such as representation of dynamics of the system represented as "SystemDynamics" <<Model>> along with the ability to model the adaptation strategies using the "Utility" <<Model>> object provided by the framework. Observation of the system state and QoS related adaptations can be made to the system using the QoS parameters represented as "Observable" and "Controllable" <<Atom>> parameters. Various controllers provided in DQME facilitate the representation of adaptation mechanisms by allowing trade-off specifications.

Practical use of DQME has been illustrated through an example model of the PCES Capstone demonstration. This case study demonstrates how DQME facilitates capturing the system dynamics and representation of the mathematical expressions. Mission requirements of the application were easily specified using DQME. Specification of the adaptation mechanisms were done using the controllers provided by DQME. Additionally, specification of tradeoffs was facilitated by providing a unidirectional connection as represented by the "Preference" connection class described in chapter 3.

Code generators have been developed that generate MATLAB and C++ code corresponding to the controllers developed in the models. The code generators have well-defined rules in the sense that the models have a concrete mapping onto the target language syntax. Automatic code-generation makes it easy to generate code for the modified models eventually saving time and eliminating the need for an additional check as there is no involvement required by other software engineer to convert the design specifications into the target code.

DQME, in short, presents the user with a control-centric perspective for the representation and analysis of software adaptation in DRE systems. As the adaptation strategies can be specified in DQME using well-defined specifications and various interactions can be shown precisely, utilizing DQME for integrating QoS in DRE systems at design-time is easier and less error-prone when compared with the other techniques discussed earlier.

Future Work

Several different enhancements can be identified for future areas of investigation. One area is the integration of a model-checking tool so as to enable formal reasoning about the adaptation mechanisms. Integration of such a tool would permit the safety and reachability analysis of the models constructed. This analysis addresses the question whether an unsafe region in the state-space is reachable by the system trajectories starting from a set of initial states [73].

Formal verification of the models constructed may be the other area that could be addressed for future enhancements to the models. Post verification process, the models could then be subjected to the code generators that would assign execution semantics to the models thus reducing the number of domain model iterations required before desired QoS levels are achieved.

APPENDIX A

GME Modeling Concepts

The GME modeling environment offers the user, the flexibility and freedom to describe a system in sufficient detail for meaningful analysis of the models. Various design choices while building models are represented by the *modeling paradigm*. A modeling paradigm is defined by the kind of models that can be built using it, how they are organized and what information is stored in them. Modeling various concepts of any particular system under consideration requires the knowledge of the basic building blocks that GME provides. The following sub-sections provide some information about these blocks which have also been used to model the DQME paradigm.

A.1 MODEL

Model is an abstract representation of an object. What a model represents depends on the domain being modeled. For instance, a process model represents functionality in a plant in the chemical engineering domain.

A model is, in computational terms, an object that can be manipulated. It has *state*, *identity*, and *behavior*. The purpose of the GME is to *create and manipulate* these models. A model typically can contain various *parts* i.e. other objects contained within the model. These parts could be one or more of the following:

- atoms (or *atomic* parts),
- other models,
- references (which can be thought of as pointers to other objects),
- sets (which can contain other parts),

- connections

In the GME, each part (atom, model, reference, or set) is represented by an icon. Parts have a simple, paradigm-defined icon. If no icon is defined for a model, it is shown using an automatically generated rectangular icon with a 3D border.

A.2 ATOM

Atoms (or *atomic parts*) are simple modeling objects that do not have internal structure (i.e. they do not contain other objects), although they can have attributes. Atoms can be used to represent entities, which are indivisible, and exist in the context of their parent model.

A.3 REFERENCE

References are parts that are similar in concept to pointers found in various programming languages. When complex models are created (containing many, different kinds of atomic and hierarchical parts), it is sometimes necessary for one model to directly access parts contained in another. For example, in one diagram of a system a variable may be defined, and in another diagram of the system one may want to use that variable.

In GME this can be attained through using – *reference.* References are objects that refer to (i.e. *point* to) other modeling objects. Thus, a reference can point to a model, an atomic part of a model, a model embedded in another model, or even another reference part or a set. A reference can be created only after the referenced part has been created, and the referenced object cannot be removed until all references to it have been removed. However, it is possible to create null references, i.e. references that do not refer to any objects. One can think of these as placeholders for future use.

A.4 CONNECTION

Relationships among objects mentioned above can be expressed using *connection*. A connection is a line that connects two parts of a model. Connections have at least two attributes: *appearance* (to aid the user in making distinctions between different types of connections) and *directionality* (as distinguished by the presence or absence of an arrow head at the "destination" end of the line). Additional connection attributes can be defined in the meta-model, depending on the requirements of the particular modeling paradigm.

A.5 ASPECT

Hierarchy is used to show or hide design detail within the models. However, large and/or complex modeling paradigms can lead to situations where, even within a given level of design hierarchy, there may be too many parts displayed at once. To alleviate this problem, models can be partitioned into *aspects*.

An aspect is defined by the kinds of parts that are visible in that aspect. The existence or visibility of an object within a particular aspect is determined by the modeling paradigm. A given object may also be visible in more than one aspect. For every kind of object, there are two kinds of aspects: primary and secondary. Objects/Parts can only be added or deleted from the model from within its primary aspect. Secondary aspects merely inherit parts from the primary aspects. Different interconnection rules may apply to parts in different aspects.

A.6 ATTRIBUTES

Models, atoms, references, sets and connections can all have *attributes*. An attribute is a property of an object that is best expressed textually. Typically objects have multiple attributes, which can be set using "non-graphical" means, such as entry fields, menus, buttons, etc. The attribute values are translated into object values (e.g. numbers, strings) and assigned to the objects. The modeling paradigm defines what attributes are present for what objects, the ranges of the attribute values among others.

A.7 INHERITANCE

GME also offers a way of representing the Object-Oriented inheritance concepts amongst the objects in a modeling paradigm. A normal object-oriented concept of Inheritance can be denoted by plain inheritance (denoted as a triangle). GME also supports two other special types of inheritances: an implementation inheritance and an interface inheritance. The details of these have been explained below.

A.7.1 INHERITANCE

The semantics of inheritance are uncomplicated: specialized (i.e. child) classes contain all the attributes of the general (parent) class, and can participate in any association the parent can participate in. However, during meta-model composition, there are cases where finer-grained control over the inheritance operation is necessary. Therefore, two types of inheritance operations between class objects—implementation inheritance and interface inheritance was introduced. The union of implementation inheritance and interface inheritance represents the normal UML inheritance.

A.7.2 IMPLEMENTATION INHERITANCE

In implementation inheritance, the subclass inherits all of the base class' attributes, but only those containment associations where the base class functions as the container. No other associations are inherited. Implementation inheritance is represented graphically by a UML inheritance icon containing a solid black dot.

A.7.3 INTERFACE INHERITANCE

Interface inheritance allows no attribute inheritance but does allow full association inheritance, with one exception: containment associations where the base class functions as the container are not inherited. Interface inheritance is represented graphically by a UML inheritance icon containing an un-colored black dot.

A.8 PROXY OBJECTS

GME provides support for Proxy objects for almost all of the concepts that can be modeled using it. Proxy objects can be treated as pointers to the actual objects. For e.g. Atom Proxy is an object that can refer an Atom modeled in the system. This enables a clean organization of objects in groups without being constrained to include all of them in the same modeling sheet.

APPENDIX B

Brief Description of Network Related Terms

B.1 JITTER

In the field of Networking, especially the IP networks, jitter refers to the variation in the delay of the data packets arrival [85]. This term is associated with the loss or de-sequencing of the data packets resulting in a delay over time from point-to-point in a network. The amount of jitter tolerable in a network is dependent on the jitter buffer available. The more the buffer, the more the network can reduce the effects of jitter.

B.2 LATENCY

Latency refers to the time taken for the data packets to arrive at the destination end. For precision, *one-way* latency is defined as the time taken from the start of data transmission to the start of packet reception [85]. The time taken from the start of packet transmission to the end of reception is often referred to as transmission delay. *Round-trip* latency is the time taken from the source transmitting packets to the source receiving a response.

B.3 THROUGHPUT

The amount of data that is transferred from one point to another or processed in a specified amount of time is referred to as throughput. The data transfer rates for networks or for disk drives are measured in terms of throughput. It is typically measured in kbps, Mbps or Gbps [84].

REFERENCES

[1] T.F. Abdelzaher, J.A. Stankovic, C. Lu, R. Zhang, and Y. Lu, "Feedback Performance Control in Software Services," IEEE Control Systems, vol. 23, no. 3, June 2003

[2] Chenyang Lu, Xiaorui Wang, Xenofon Koutsoukos, "End to End Utilization Control in Distributed Real-Time Systems," ICDCS 2004, IEEE Computer Society 2004, 456 466

[3] Chenyang Lu, Member, IEEE, Xiaorui Wang, and Xenofon Koutsoukos, "Feedback Utilization Control in Distributed Real-Time Systems with End-to-End Tasks," IEEE Transactions on Parallel and Distributed Systems, vol. 16, no. 6, June 2005

[4] www.wikipidea.org – Article on "Control Theory"

[5] A. Burns, A. Wellings. Real-Time Systems and Programming Languages, 3rd Edition. Addison Wesley Longmain, 2001

[6] C. Lu, J. Stankovic, G. Tao, and S. Son, "Feedback Control Real-Time Scheduling: Framework, Modeling, and Algorithms," J. Real-Time Syst., vol. 23, no. 1-2, pp. 85-126, July/September, 2002

[7] T. F. Abdelzaher, K. G. Shin, and N. Batti, "Performance Guarantees for Web Server End-Systems: A Control Theoretic Approach," IEEE Trans. Parallel & Dist. Syst., vol. 13, no. 1, pp. 80-96, January 2002

[8] T. F. Abdelzaher and N. Bhatti, "Web Server QoS Management by Adaptive Content Delivery," International Workshop on Quality of Service, 1999

[9] Ledeczi A., Maroti M., Bakay A., Karsai G., Garrett J., Thomason IV C., Nordstrom G., Sprinkle J., Volgyesi P.: The Generic Modeling Environment, Workshop on Intelligent Signal Processing, accepted, Budapest, Hungary, May 17, 2001

[10] GME Manual and User Guide - http://www.isis.vanderbilt.edu/Projects/gme/GMEUMan.pdf

[11] J. Sztipanovits, G. Karsai: Model-Integrated Computing, IEEE Computer, pp. 110-112, April, 1997

[12] www.quo.bbn.com

[13] David Tacconi, Cem Saraydar, Şirin Tekinay "Ad hoc enhanced routing in UMTS for increased packet delivery rates," Wireless Communications and Networking Conference, 2004. WCNC, 2004 Vol 2. Pg – 1093-1098

[14] Sangyoon Oh1,2, Hasan Bulut1,2, Ahmet Uyar1,3, Wenjun Wu1, Geoffrey Fox1,2 “Optimized Communication using the SOAP Infoset For Mobile Multimedia Collaboration Applications,” Proceedings of the International Symposium on Collaborative Techniques and Systems. May 2005, Missouri, USA

[15] Object Management Group, Realtime CORBA Joint Revised Submission, OMG Document orbos/99-02-12 ed., March 1999

[16] Morgenthal JP 1999 Microsoft COM+ Will Challenge Application Server Market www.microsoft.com/com/wpaper/complus-appserv.asp

[17] Wollrath A, Riggs R and Waldo J 1996 A Distributed Object Model for the Java System. USENIX Computing Systems

[18] James R. Davis, “Model Integrated Computing: A Framework for Creating Domain Specific Design Environments “

[19] Zonghua Gu, Shige Wang, Sharath Kodase and Kang G. Shin “An End-to-End Tool Chain for Multi-View Modeling and Analysis of Avionics Mission Computing Software,” 24th IEEE International Real-Time Systems Symposium (RTSS'03) p. 78

[20] Gray, J., Bapty, T., Neema, S., Ledeczi, A.: "Viewpoints and Aspects in Domain-Specific Modeling," 1st International Conference on Aspect-Oriented Software Development, Demonstration, Enschede, The Netherlands, April 2002

[21] Sriram Narasimhan, Gautam Biswas, Gabor Karsai, Tal Pasternak, and Feng Zhao Building observers to address fault isolation and control problems in hybrid dynamic systems, Proceedings of the IEEE SMC 2000 Conference, 2000

[22] Peter Volgyesi, Akos Ledeczi. “Component-Based Development of Networked Embedded Applications,” 28th Euromicro Conference, Sep 2002

[23] Larry Howard “Adaptive Learning Technologies for Bioengineering Education,” IEEE Engineering in Medicine and Biology Magazine

[24] Christopher D. Gill, David L. Levine Douglas C. Schmidt “Towards Real-Time Adaptive QoS Management in Middleware for Embedded Computing Systems,” Fourth Annual Workshop on High Performance Embedded Computing, MIT Lincoln Laboratory, 2000

[25] Schantz, and A. K. Atlas, “Applying Adaptive Real-time Middleware to Address Grand Challenges of COTS-based Mission-Critical Real-Time Systems,” in Proceedings of the 1st IEEE International Workshop on Real-Time Mission-Critical Systems: Grand Challenge Problems, Nov. 1999

[26] http://www.isis.vanderbilt.edu/research/mic.html

[27] Earl Long, Amit Misra, and Janos Sztipanovits, "Increasing Productivity at Saturn," IEEE Computer, August 1998, pp. 35-43

[28] J. Rumbaugh, I. Jacobson, and G. Booch, "The Unified Modeling Language Reference Manual," Addison-Wesley, 1998

[29] Barr, Michael "Closed-Loop Control," Embedded Systems Programming, August 2002, pp. 55-56

[30] Thomas Kailath "Linear Systems," Prentice Hall, Englewood Cliffs N.J 1980 Pg. 262

[31] http://www.objs.com/survey/QoS.htm

[32] Nanbor Wang, Christopher Gill, Douglas Schmidt, Aniruddha Gokhale, Balachandran Natarajan, Joseph Loyall, Richard Schantz, and Craig Rodrigues. "QoS-enabled Middleware," Chapter in Middleware for Communications, Qusay H. Mahmoud (Editor), Wiley, July 2000

[33] A. Gokhale and D. C. Schmidt, "Techniques for Optimizing CORBA Middleware for Distributed Embedded Systems," in Proceedings of INFOCOM '99, Mar. 1999

[34] Atif Memon, Adam Porter, Douglas Schmidt "Feedback-driven Design of Distributed Real-time & Embedded Component Middleware Via Model-Integrated Computing & Distributed Continuous Quality Assurance," Abstract - http://www.cs.virginia.edu/~sullivan/sdsis/Program/Adam%20Porter.pdf

[35] Object Management Group "The Common Object Request Broker: Architecture and Specification," 2.3 ed., June 1999

[36] Sun Microsystems, "Jini Connection Technology," http://www.sun.com/jini/index.html, 1999

[37] Anne Thomas, Patricia Seybold Group, "Enterprise JavaBeans Technology," http://java.sun.com/products/ejb/white paper.html, Dec. 1998. Prepared for Sun Microsystems, Inc

[38] D. Box "Essential COM," Addison-Wesley, Reading, MA, 1997

[39] IBM, "MQSeries Family," http://www-4.ibm.com/software/ts/mqseries/, 1999

[40] Rodrigues C. "Using Quality Objects (QuO) Middleware for QoS Control of Video Streams," OMG Embedded and Real-Time Distributed Object Systems Workshop. January 7-10, Burlingame, CA

[41] N Wang, DC Schmidt, M Kircher, K Parameswaran "Towards a Reflective Middleware Framework for QoS-enabled CORBA Component Model Applications," IEEE Distributed Systems Online, 2001

[42] Bollella, Gosling, Brosgol, Dibble, Furr, Hardin, and Turnbull, “The Real-Time Specification for Java,” Addison-Wesley, 2000

[43] Object Management Group “Dynamic Scheduling Real-Time CORBA Joint Revised Submission,” OMG Document orbos/2000-08-12 ed., August 2000

[44] DARPA, “The Quorum Program.” http://www-fp.mcs.anl.gov/dsl_internal/collaboration/Quorum.html, 1999

[45] Richard Schantz, Joseph Loyall, Michael Atighetchi, Partha Pal, “Packaging Quality of Service Control Behaviors for Reuse,” 5th IEEE International Symposium on Object-Oriented Real-time distributed Computing, April 29 - May 1, 2002, Washington, DC

[46] Pal PP, Loyall JP, Schantz RE, Zinky JA, Shapiro R, Megquier J. “Using QDL to Specify QoS Aware Distributed (QuO) Application Configuration,” Proceedings of ISORC 2000, 3rd IEEE International Symposium on Object-Oriented Real-time distributed Computing, March 15 - 17, 2000, Newport Beach, CA

[47] Vanegas R, Zinky JA, Loyall JP, Karr DA, Schantz RE, Bakken DE “QuO's Runtime Support for Quality of Service in Distributed Objects,” Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'98), 15-18 September 1998, The Lake District, England

[48] R. Schantz, J. Loyall, C. Rodrigues, D.C. Schmidt, Y. Krishnamurthy, and I. Pyarali “Flexible and Adaptive QoS Control for Distributed Real-time and Embedded Middleware,” The ACM/IFIP/USENIX International Middleware Conference, June 2003, Rio de Janeiro, Brazil

[49] J. Loyall, R. Schantz, J. Zinky, P. Pal, R. Shapiro, C. Rodrigues, M. Atighetchi, D. Karr, J.M. Gossett, and C.D. Gill. “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

[50] J. Loyall, D. Bakken, R. Schantz, J. Zinky, D. Karr, R. Vanegas, and K. Anderson. “QoS Aspect Languages and Their Runtime Integration,” Lecture Notes in Computer Science, 1511, Springer-Verlag. Proceedings of the Fourth Workshop on Languages, Compilers, and Run-time Systems for Scalable Computers (LCR98), May 28-30, 1998

[51] J. Loyall, R. Schantz, J. Zinky, and D. Bakken. “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

[52] L. R. Welch, B. Ravindran, B. Shirazi and C. Bruggeman, “Specification and analysis of dynamic, distributed real-time systems,” in Proceedings of the 19th IEEE Real-Time Systems Symposium, 72-81, IEEE Computer Society Press, 1998

[53] Ryan Detter, Lonnie R. Welch, Barbara Pfarr, Brett Tjaden, and Eui-Nam Huh, “Adaptive management of computing and network resources for spacecraft systems,” The 3rd Annual Military and Aerospace Applications of Programmable Devices and Technologies Conference (MAPLD 2000), September 2000

[54] Lonnie R. Welch and Behrooz A.Shirazi “A Distributed Architecture for QoS Management of Dynamic, Scalable, Dependable, Real-Time Systems,”

[55] J. Huang, R. Jha, W. Heimerdinger, M. Muhammad, S. Lauzac, B. Kannikeswaran, K. Schwan, W. Zhao and R. Bettati, “RT-ARM: A real-time adaptive resource management system for distributed mission-critical applications,” in Workshop on Middleware for Distributed Real-Time Systems, RTSS-97, (San Francisco, California), IEEE, 1997

[56] Benedikt Paats “Generalized Quality of Service: QoS from the System Architecture Perspective,” Seminar – Communication and Multimedia, 2005

[57] C. L. Phillips and H. T. Nagle. Digital Control System Analysis and Design (3rd edition). Prentice Hall, 1995

[58] A. Burns, A. Wellings “Real-Time Systems and Programming Languages,” 3rd Edition. Addison Wesley Longmain, 2001

[59] C. Lu, G. A. Alvarez, and J. Wilkes. Aqueduct: “Online data migration with performance guarantees,” In Proc. USENIX Conf. File Storage Tech., pages 219–230, 2002

[60] S. Qin and T. Badgewell “An overview of industrial model predictive control technology,” Chemical Process Control, 93(316):232–256, 1997

[61] S. Parekh et al., “Using Control Theory to Achieve Service Level Objectives in Performance Management,” J. Real-Time Syst., vol. 23, no. 1-2, pp. 127-141, July/September 2002

[62] S. Mascolo “Classical Control Theory for Congestion Avoidance in High-Speed Internet,” Proc. Conf. Decision & Control, pp. 2709-2714, 1999

[63] E. Marshall “Control of time-delay systems,” Peter Peregrinus Ltd, 1979

[64] J. Astrom, B. Wittenmark “Computer controlled systems**,”** Prentice Hall, Englewood Cliffs, N. J., 1984

[65] Z. Lu et al. “Control-Theoretic Dynamic Frequency and Voltage Scaling for Multimedia Workloads,” Proc. Int’l Conf. Compilers, Architectures, & Synthesis Embedded Syst. (CASES), pp. 156-163, 2002

[66] P. Antsaklis, editor. “Special Issue on Hybrid Systems,” Proceedings of the IEEE. July 2000

[67] P. Antsaklis, X. Koutsoukos, and J. Zaytoon "On hybrid control of complex systems: a survey," European Journal of Automation, 32:1023–1045, 1998

[68] J. Gray, S. Neema et. al. "Concern Separation for Adaptive QoS Modeling in Distributed Real-Time Embedded Systems," http://gray-area.org/ Publications section – Submitted, Under Review

[69] Harel, David, "Statecharts: A Visual Formalism for Complex Systems", Science of Computer Programming, 8, 1987, pp. 231-274

[70]] http://www.mathworks.com Product: MATLAB

[71] http://cnx.rice.edu/content/m2101/latest/

[72] Derek Rowell "State-Space Representation of LTI Systems" Oct 2002

[73] Stephen Prajna, Anders Rantzer " Primal–Dual Tests for Safety and Reachability,"s Lecture Notes in Computer Science, Volume 3414, Jan 2005, Pages 542 – 556

[74] Networked Multimedia II End-To-End Quality-of-Service (QoS) - http://nrg.cs.usm.my/~tcwan/Notes/MM-Ntwk-II.doc

[75] Definition of QoS - http://compnetworking.about.com/od/networkdesign/l/bldef_qos.htm

[76] Article on "IntServ" or Integrated Services Architecture - http://en.wikipedia.org/wiki/Integrated_services

[77] Article on "DiffServ" or Differentiated Services Architecture - http://en.wikipedia.org/wiki/Differentiated_services

[78] Sherif Abdelwahed, Nagarajan Kandaswamy, Sandeep Neema " Online Control for Self-Management in Computing Systems," 10th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'04) May 25 – 28, 2004 Toronto, Canada

[79] Sujata Mujumdar, Nag Mahadevan, Sandeep Neema, Sherif Abdelwahed "A Model-Based Design Framework to Achieve End-To-End QoS Management," 43rd ACM Southeast Conference, March 18-20, 2005, Kennesaw, GA, USA

[80] DSRAS Technical Report

[81] Stateflow Semantics - http://www.mathworks.com/access/helpdesk/help/toolbox/stateflow/ug/f26-1032049.html

[82] Model Weaver - http://www.isis.vanderbilt.edu – Projects – PCES

[83] PCES Demo Presentations - http://www.isis.vanderbilt.edu – Projects – PCES – Presentations

[84] www.webopedia.com – Terms: throughput

[85] www.wikipedia.com – Terms: Jitter, Latency