Experiences Using Adaptive Middleware in Distributed Real-time Embedded Application Contexts: a Dependability Perspective

Christopher D. Gill
Washington University,
St. Louis

Joseph P. Loyall, **Richard E Schantz**
BBN Technologies,
Cambridge

Douglas C. Schmidt
University of California,
Irvine

Abstract

Over the past few years we have developed a number of real world applications that have both motivated and used middleware technologies for COTS-based distributed object computing in general, and runtime adaptive behavior in particular. In this context, a new twist on the theme of dependability involves adaptation as an effective means for dealing with the less-than-optimum situations that often arise due to failures and other forms of sudden, unexpected behavior. This paper briefly describes two of these applications, Weapons Systems Open Architecture (WSOA) and Unmanned Aerial Vehicles (UAV), which have already undergone a series of evaluation steps to determine the suitability of their concepts and implementations under realistic usage scenarios.

Our focus in this paper is on the adaptivity exhibited by WSOA and UAV applications under changing operating conditions, and on the technical basis for the effective marshaling of modified workplans to keep application mission objectives focused using available resources. We summarize some of the lessons learned thus far in developing these applications and the underlying middleware technologies, and show how they influence each other. We also assess the suitability of the solutions offered, discuss some of the difficulties encountered along the way, and outline the means we are applying to overcome them. We conclude with some of the key research challenges that must be resolved to field distributed real-time and embedded systems that can be depended upon to perform adequately, even under extreme and unusual operating circumstances.

1. Introduction

1.1. Emerging Trends

The next-generation of mission-critical distributed real-time and embedded (DRE) systems require an increasingly wide range of features, while at the same time minimizing costs. For instance, next-generation avionics mission computing systems [WSOA] must collaborate with remote command and control systems, provide on-demand browsing capabilities for human operators, and respond flexibly to unanticipated situational factors that arise in run-time environments [UAV]. Moreover, these systems must perform unobtrusively, shielding human operators from unnecessary details, while simultaneously communicating and responding to mission-critical information at an accelerated operational tempo.

The characteristics of next-generation DRE systems present QoS requirements for shared resources and workloads that can vary significantly at run-time. In turn, this increases the demands on end-to-end system resource management and control, which make it hard to simultaneously (1) create effective resource managers using traditional statically constrained allocators and schedulers, (2) achieve reasonable resource utilization, and (3) meet the individual application needs for tradeoff preferences. In addition, the mission-critical aspects of these systems require that they respond adequately to both anticipated and unanticipated operational changes in their run-time environment, and ensure critical components can acquire available resources. Meeting the increasing QoS demands of next-generation real-time systems motivates the need for adaptive middleware-centric abstractions and techniques, such as the automated reconfiguration, layered resource management, and dynamic scheduling techniques focused on in our research.

Introducing application-level awareness of changes to expected and delivered QoS is a new direction for inserting adaptive behavior into DRE applications. Adaptation can occur at any and all of the various layers of the system, including customized approaches in the application itself and standard service (re)configurations within the supporting middleware infrastructure. An example of an application-level adaptation might be moving from full motion video to audio and still imagery to text-only interactions. An example of a service-level adaptation might be acquiring additional bandwidth by preempting lower priority users or automatically instantiating additional resource replicas when another one becomes unreachable. The key to success in these adaptations lies in developing paths through the system layers that can effectively coordinate the otherwise independent activities so they provide maximum utility to developers and users without conflicting behavior that might otherwise result from a series of independent or transparent actions.

Computer system dependability has often traditionally meant, and continues to mean, redundancy management in

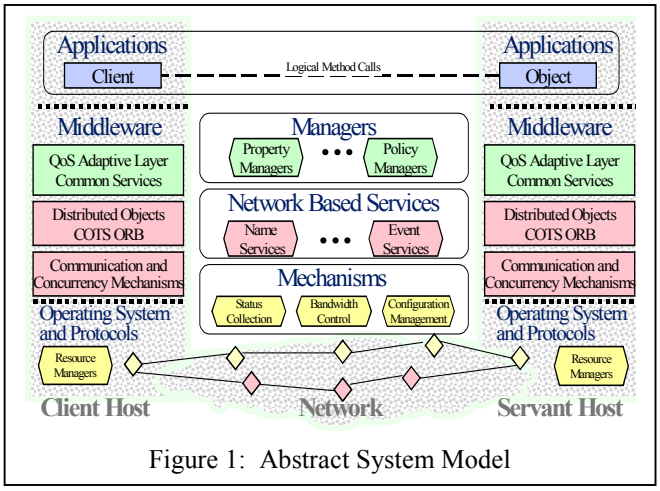

Figure 1: Abstract System Model

general and replication management in particular. In that tradition, an application or system has some specific fixed functionality. Moreover, there are parts of the system that when integrated together and behaving properly with adequate resources provide that functionality. By analyzing the potential failures and providing enough redundancy, the idea is generally to mask the fact that a failure has occurred by substituting failed ones with alternate parts that provide roughly the same capability. There has been considerable success with that approach, but also significant limitations, especially as the systems we try to make dependable become more distributed and complex, and are subject to many more variations of not only failures but also changes in the operating environment.

It is particularly hard to mask anomalies in DRE systems whose dispersed components also have real time performance constraints. A complementary approach to this type of dependability management is providing the best service under the circumstances, even when those circumstances may not include all of the desirable attributes of the intended computation or system. A truly dependable system can be expected to do the "right thing" under whatever circumstances arise and whatever resources are available. It is this type of dependability that we explore in this paper, i.e., *the behavior of a system under unexpected or unplanned conditions*. Our goal is to provide the most effective behavior under any resource availability circumstances. Our approach is to alter the pattern of interactions or simultaneously modify the application or system behavior itself to an alternate form more suited to the circumstances, instead of–or in addition to–having identical replicated services available.

1.2. A Common Middleware Framework for Adaptable Distributed Computing

Figure 1 illustrates our general concept of middleware and some of the key layers we are using to organize our technology integration activities. Based on our prototyping and benchmarking activities to date [ICDCS], these integrated components enable an unprecedented degree of application-level control and adaptability to varying conditions typically found in both embedded and Internet environments. In turn, these integrated capabilities enable a new generation of DRE applications whose resource management strategies can be customized readily, without the need to make application developer responsibilities significantly more complex or risky.

2. DRE Examples

This section discusses two example DRE applications we've built, evaluated, and experimented with as part of our investigation of managing QoS and adaptive behavior to meet mission requirements under varying and changing operating conditions. The first of these (WSOA) is part of a fielded concept of operation in conjunction with the Boeing Company and AFRL supporting dynamic inflight replanning. The second (UAV) represents a real-time video sensor capture, dissemination, and processing capability with feedback, linking mobile and fixed assets, operating over shared resources.

2.1. Weapons System Open Architecture (WSOA)

The Weapons System Open Architecture (WSOA) program has explored the possibility of building an open-systems testbed environment in which legacy embedded systems in the avionics domain can be integrated within a next-generation middleware quality of service (QoS) management context to provide unprecedented capabilities for time-critical target prosecution. The major feature areas provided by the WSOA framework (see Figure 2) are as follows:

- Tactical communication links, using legacy Link 16 technology to connect mission computers on an F-15 aircraft with imagery terminal servers on a command-and-control (C2) aircraft.

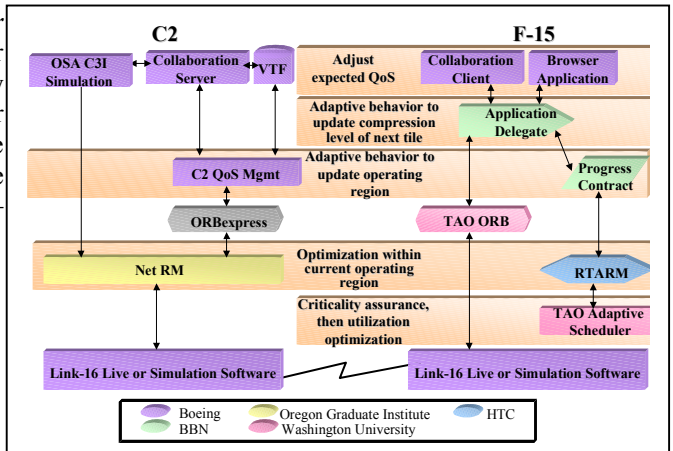

Figure 2: WSOA dynamic replanning system architecture

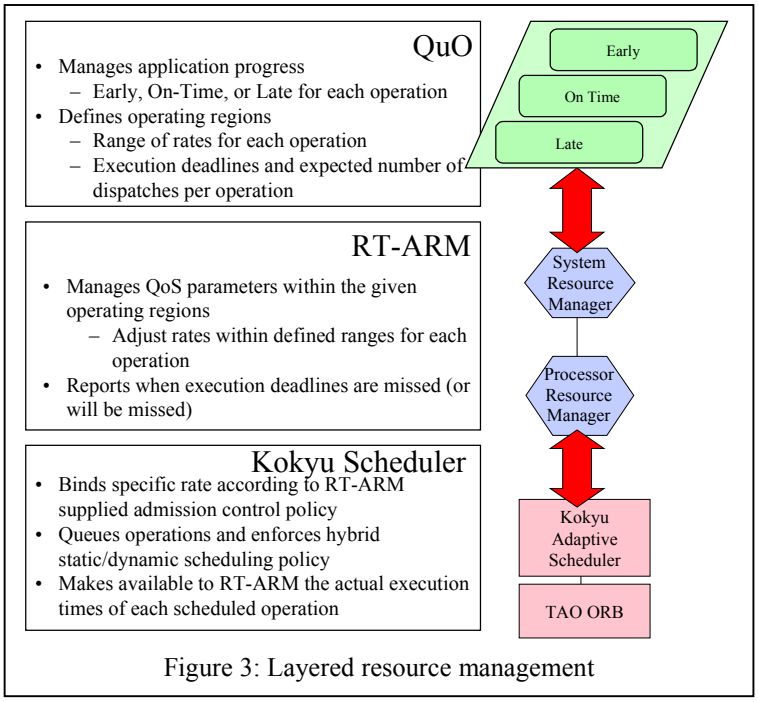

Figure 3: Layered resource management

- Collaborative planning between F-15 and C2 personnel during a mission.
- Giving F-15 personnel information-mining capabilities on the C2 imagery libraries, via a browser-like interface using standard F-15 cockpit equipment.
- Link bandwidth optimization, using contract-based evaluation of application requirements and system resources and loads.
- Adaptive and dynamic QoS management, to ensure maintenance of critical assurances, while optimizing and tuning non-critical performance.

This paper highlights the last two feature areas: link bandwidth optimization and adaptive and dynamic QoS management. To provide assurances and optimization of key QoS system properties, coordinating diverse approaches to a number of QoS management areas is fundamental. In particular, we have integrated several forms of advanced middleware capabilities within the WSOA testbed, including the following technologies that constitute a layered middleware architecture (from top to bottom; see Figure 3:

- QuO[QuO], an end-to-end QoS management middleware framework developed at BBN Technologies
- RT-ARM[RT-ARM], a real-time adaptive resource manager developed at Honeywell Technologies
- Kokyu[Kokyu], a real-time scheduling and dispatching framework developed at Washington University in St. Louis
- TAO[TAO], a high-performance and real-time CORBA-compliant object request broker (ORB) developed at Washington University in St. Louis and the University of California, Irvine.

In this section we consider how the WSOA project provides insights into the nature of dependability in large-scale heterogeneous systems with demanding QoS requirements. Of particular interest is how the balance of (1) strict assurances for critical system behavior and (2) adaptive tuning and optimization of non-critical behavior is achieved. Each middleware layer addresses separate concerns, and yet those concerns must be woven seamlessly end-to-end and layer-to-layer to achieve robust and dependable system performance in complex mission-critical systems, such as the WSOA testbed framework.

Robust behavior during overload. One key area in which dependability must be considered is whether skillful management of resources can allow a more robust response to scarcity in worst-case conditions, as well as giving better overall performance in the average case. In WSOA, real-world constraints on power consumption, weight, and frequency of processor upgrades, limit the ability to offer sufficient resources through excessive over-provisioning. For example, the complexity of the environment in which WSOA is expected to operate, combined with the further complexity of systems-of-systems integration end-to-end, means that explicit testing of all possibilities is not possible. Instead, other techniques, such as combining testing and model-based analysis must be pursued [Stuart:01].

While the *total* system requirements for the WSOA testbed are both high in quantity and variable in the face of environmental variations, a key insight is that systems of this kind may still be engineered so that a *critical subset* of the total requirements can be implemented so that they are both (1) lower in quantity and (2) more predictable in both time and in interaction with other system features. For example, operations can be designed in a way similar to the imprecise computations model [ChungLiuLin:90] in that a fixed *sufficient* number of computation steps could be performed by a single critical operation, with additional *desirable* computation steps being performed by a sequence of non-critical operations whose execution is not assured. This means that resources must be provisioned to assure the maximum *critical* requirements can be met, and that additional processing be managed dynamically and adaptively with remaining resources.

The changing nature of proper functioning. In an avionics mission-computing environment, such as WSOA, operations may exhibit different degrees of criticality, depending on the environmental and system context. For example, consider computing the next segment of a navi-

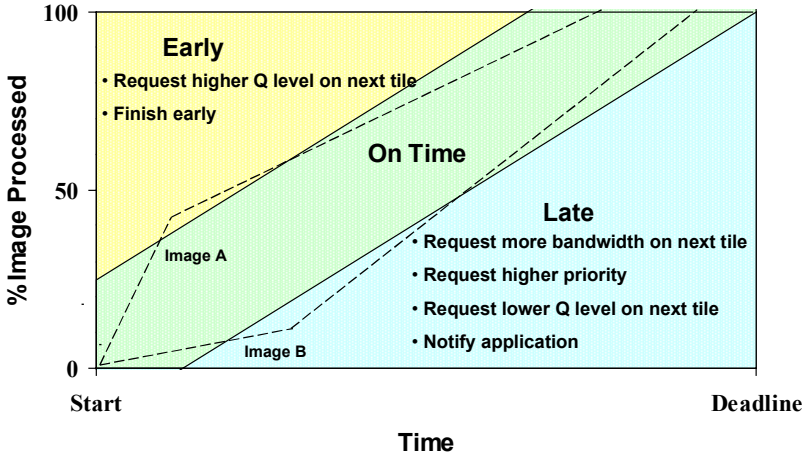

Figure 4: A QuO contract monitors the progress of image download and adapts to keep it within the "On Time" range

gation route, with each segment going between two waypoints. In some situations, such as under low-threat conditions, the first two segments from the aircraft's current position may be the only ones considered critical, and others can be designated non-critical to reduce the demands of the critical subset of processing. In other situations, such as egress from a high-threat environment, more route planning may be considered critical, so that additional navigation operations must be added to – and presumably other kinds of processing can be removed from – the critical subset.

Interestingly, non-critical processing may also be made more dependable through context-aware adaptive management. For example, if a non-critical navigation route segment cannot be computed on time, the application might retry the computation rather than proceed to compute the next segment, whose origin would depend on the one that was not computed. As a similar example, imagery is downloaded as a sequence of tiles of varying quality, radially from a point of interest in the image. WSOA uses the QuO middleware to monitor download progress, and tune image tile compression upward as necessary to fit within the specified image transmission time (see Figure 4). Clearly, image tiles near the point of interest must be kept at as low compression as possible to improve image quality – however, additional surrounding context in the image may still be of sufficient resolution at lower quality to be useful.

The issue of timeliness for imagery download is interesting in the WSOA context due to the complexity of the libraries themselves, and the degree to which compression levels depend on the contents of the images themselves. While a QuO contract may request a level of compression matching expectations of timely download and sufficient quality, the image itself may result in discrete alternative compression levels for each tile. Moreover, system load, user inputs, and other factors may further perturb the conditions under which QoS is managed.

A control-centric QoS management architecture is therefore necessary to maintain dependable system behavior in the face of unanticipated conditions. QuO monitors and adjusts its perception of system behavior regularly, so that it maintains a clear picture of the *actual* conditions under which it is controlling system QoS. RTARM monitors a different set of conditions, such as whether it is succeeding in meeting control boundaries for processing (decompression, storage, and delivery) tiles that arrive at the F-15 endsystem, and when it cannot feasibly schedule that processing either directs Kokyu to reschedule the CPU with some operations at lower ranges of available rates, or reports back to QuO that it cannot meet its obligations and QuO can respond accordingly. What is most important here is that the control loop is *closed*, even though it crosses multiple QoS management layers.

Moving design and resource management decisions to later in the software engineering cycle. In the Bold Stroke domain-specific middleware, upon which the WSOA framework is based, late binding was applied initially to software, in an evolution from hand-crafted assembly language, to an object-oriented component-based CORBA approach written in C++. The effect of this evolution was that the point in the software lifecycle at which functionality could be applied was pushed significantly later. For example, in the hand-crafted version, all functionality had to be defined before the design of a cyclic executive that ran the components. In the initial CORBA version, rate monotonic scheduling was done at system build time, and through careful design of system modes, the set of objects that was active could be selected at run-time.

WSOA takes this idea of late binding to another level, particularly in the area of QoS management. Hybrid static/dynamic scheduling and adaptive selection of execution rates was applied by Kokyu to defer operation scheduling until run-time. The RTARM performed adaptive monitoring and adjustment of available rates to maintain system operating conditions within specified limits. QuO used contract-based monitoring and evaluation to integrate diverse resource management techniques (*e.g.,* image compression levels and rates of execution for the decompression operation) and ensure QoS requirements were met end-to-end.

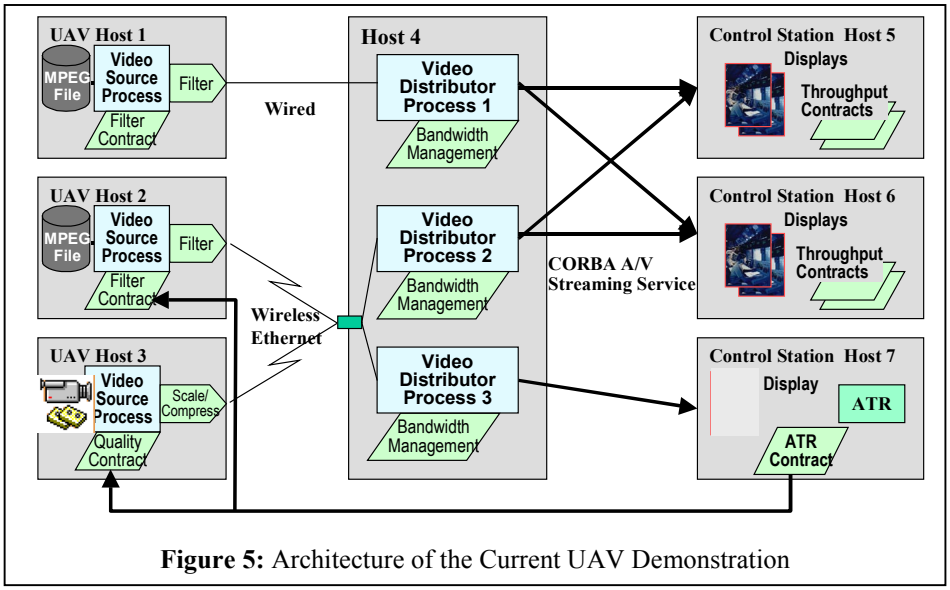

Figure 5: Architecture of the Current UAV Demonstration

Knitting the different perspectives into a unified management capability. The various capabilities described above need to be integrated together to form a cohesive system-level solution, which involves information sharing across and between these views. Each of the three levels of adaptive QoS management described above, QuO, RTARM, and Kokyu, operates on a different time scale. QuO can make image compression adjustments on a per-tile basis, with a maximum rate on the order of seconds. RTARM also bases its decisions on rate selection, and its maximum responsiveness is therefore also on the order of seconds. Note that QuO might make both a compression level decision and handle a report of failure to maintain the given QoS limits from RTARM in the same interval, though such case are expected to be infrequent. The Kokyu scheduler makes operation scheduling decisions whenever requested by RTARM, so its rate is naturally tied to the RTARM. As noted below, however, Kokyu is designed to minimize the fraction of that interval that it uses, to minimize overhead of adaptive responsiveness and improve the *possible* responsiveness, *e.g.*, if the RTARM were calibrated to run at a faster rate. Finally, the Kokyu dispatcher makes dynamic scheduling decisions and multiplexes dispatch requests onto prioritized threads, on the order of ten microseconds. As a result, as little overhead is added to the in-band (as compared to Kokyu scheduling, RTARM, and QuO adaptation, which operate out-of-band) path as possible.

Experimentation. To measure the benefits of multi-layer adaptive management, we are in the process of conducting experiments within a realistic F-15 OFP hardware and software environment. We have instrumented the adaptation paths through the application to collect time stamps around the following activities:

1. QuO contract evaluation
2. QuO-triggered adaptation of compression levels
3. QuO-triggered adaptation of available rates of execution for image tile processing operations
4. RTARM in-region evaluation
5. RTARM-triggered adaptation of available rates for image tile processing operations and
6. Kokyu scheduler adaptive rescheduling of operations.

In addition, we have instrumented the image tile request-download-decompression-delivery path to assess the impact of adaptive management on application performance. We will run three major experiments, with repeated trials of each: (a) full compression, without adaptation, (b) no compression, without adaptation , and (c) variable compression, using adaptation.

By obtaining and analyzing these data, we will achieve a clear profile of the following key factors for multi-layer adaptive resource management:

1. Coupling of layers in time and in overhead
2. Interactions of adaptation at different time-scales
3. Overhead measures for adaptation at each layer

Impact of adaptation on application performance – particularly how adaptation provides acceptable quality and timeliness of imagery, compared to over-compressed or under-compressed approaches without adaptation.

2.2. The Unmanned Aerial Vehicle (UAV) Application

A second example we've worked with is an unmanned aerial vehicle (UAV) demonstration. As part of an activity for the US Navy, DARPA, and the US Air Force, we have been developing a DRE system centered on the concept of information dissemination from, and control of, UAVs. In this application, we are concerned with managing the QoS requirements for (a) the delivery of video from UAVs to a command and control (C2) platform (e.g., a shipboard environment, ground station, or air C2 node) and (b) the delivery of control signals from the C2 platform back to the UAVs. This QoS management includes trading off video quality and timeliness, and coordinating resource usage from end-to-end and among competing UAVs, to satisfy changing mission requirements under dynamic, and potentially hostile, environmental conditions.

UAV architecture. Figure 5 illustrates the architecture of the demonstration. It is a three-stage pipeline, with simulated UAVs or live UAV surrogates (such as airships with mounted cameras) sending video to processes (*dis-*

tributors) that distribute the video to the proper control stations. The UAVs in our prototypes are implemented by two different types of processes:

1. The first reads MPEG video from a file simulating high frame rate video capture devices.
2. The second is capturing video from a live camera feed, which adds realism to the prototype and provides the ability to manipulate the raw video, but produces frame rates limited by the camera's capabilities.

Our prototype also uses both wired and wireless Ethernet connections to simulate the data links from the UAVs to the distributor host. The wireless links from the second and third UAV surrogates contend for the same wireless Ethernet connection and provide a forum for experimenting with wireless video adaptation strategies. The wired Ethernet connection provides a higher bandwidth connection simulating current and emerging higher-capacity wireless transports.

The video distributor processes send the video streams to control stations on a land- or ship-based network. The control stations include video display processes and other video processing applications (the current demonstration includes an automatic target recognition – ATR – application), each with their own mission requirements.

UAV adaptivity strategies. Adaptation is used as part of an overall system concept to provide *load-invariant performance*. The video displays throughout the ship must present the current images observed by the UAV with acceptable fidelity, regardless of the network and host load, in order for the control station operators to achieve their missions, e.g., flying the UAV or tracking a target. To accomplish this, QuO system condition objects monitor the frame rates and the host load on the distributor and video display hosts. As the frame rate declines and/or the host loads exceeds a threshold, they cause region transitions, which trigger the following adaptations:

- If the network is the constrained resource, send a reduced amount of data, for example by dropping frames or compressing the video at the sender or the distributor.
- If the land-based network is the constrained resource, use a bandwidth reservation protocol to ensure that the distributor is able to send the necessary data to the viewers through the network, even when the network is congested.
- If the distributor CPU is the constrained resource, move the distributor to a different host where more resources are available.

In addition, the ATR and other video processing applications must receive a sufficient frame rate and amount of critical data content to fulfill their missions, e.g., to recognize threats and targets in the video images. To accomplish this, QuO contracts on the UAV video sources coordinate to share the wireless link. For example, the current prototype scales and compresses the video destined for the ATR process while it filters those destined for human display as described above. This maintains a high rate of frames with sufficient information (but not necessarily enough for viewing) for the ATR and a smooth, viewable video for the human operators.

Adaptation is also used to respond to changing mission requirements and to dynamic conditions. For example, in the current UAV prototype, when the ATR process recognizes a target, it changes the mission requirements of the corresponding UAVs. Whereas previously they only needed to make sure a high rate of the critical data needed by the ATR made it through, once a target has been detected the UAV must provide human viewable video so that a commander can make a decision, an operator can track the target, etc. To accomplish this, a contract associated with the ATR host reacts to target recognition by propagating that information upstream to contracts on the UAV sources. The contracts on the UAV coordinate to achieve the new mission, i.e., providing high quality video at a high rate from the targeting UAV, in the following manner:

- The targeting UAV shuts off its scaling and compression of the video
- To accommodate the greater amount of data on the wireless link from the UAV that spotted the target, the other UAVs will reduce their frame rate to the minimal acceptable rate.

The current prototype includes diversity in the video format (MPEG and PPM), network transport (wireless, LAN, and WAN), and mission requirements (video viewing and ATR) to support a number of experiments in dynamic adaptation. The TAO A/V Streaming Service provides the flow connection between the various processes. This is an implementation of the CORBA A/V Streaming Service [CORBA-AV:97], which supports multimedia applications. The CORBA A/V Streaming Service uses RSVP as its bandwidth reservation mechanism.

Robust behavior under overload. Similarly to the WSOA application, the UAV prototype has requirements for dependability even in the presence of load or variations in the availability of resources. Also similarly to WSOA, these include both network resources and CPU resources. The C2 components (whether shipboard or ground-based) that are the target for the UAV video data do not typically face the same constraints on power consumption and weight as the WSOA avionics platform, although the UAVs themselves will frequently face even more strict constraints in these areas. Moreover, the current instantiation of the UAV prototype requires resource management in two dimensions:

1. Horizontal, or *end-to-end*, to achieve requirements of video delivery from any particular UAV to the control stations that use it (and control signals from the control stations back to the UAVs).
2. Vertical, or *coordinated*, to mediate the conflicting requirements of multiple UAVs, multiple distributors, and

multiple control stations in any stage of the pipeline, competing for a set of resources, e.g., CPU, network, and data.

The changing nature of proper functioning. In the UAV platform, mission requirements can change from control station to control station and from moment to moment, based upon environmental conditions. In normal operation, certain control station functions, such as piloting the UAV, may take priority over other functions, such as non-critical observation. The presence of threats or targets, however, might make the priorities of functions change rapidly. Context-aware adaptation can trade off less important functionality to maintain the requirements of important functionality. For example, we might choose to scale back the video frame rate from a UAV to minimize the latency of the video for a control station that is piloting the UAV. A similar adaptation can be used to maximize the data content of the particular video frames for a targeting officer who must have a high-fidelity image to make command decisions.

The conflicting demands of functional requirements can limit the types of adaptations that are possible and the places at which they occur. For example, if the piloting control station above triggers frame filtering at a UAV source to achieve lower latency, it can affect other stations that are counting on high data content, such as a collector of surveillance data for off-line analysis. A better strategy might be to reserve network resources so that both stations can achieve their requirements: *low latency and high data content*. While some adaptation decisions can be made based only on local information, mission-wide strategies must constrain the adaptation choices available. This is why it is crucial for these adaptation strategies to be programmed in middleware, where they can consider both application-level requirements and system-level mechanisms.

Separation of concerns and late binding of adaptation strategies. The UAV prototype application was developed using the QuO middleware to separate the functional concerns of the application from the adaptation, QoS, and dependability concerns. This enabled the functional behavior of the application – video capture, distribution, display, and control – to be developed without entangling information about the platform in which the application will be hosted. The development of the UAV "system" then becomes a "construction" process, combining functional components and QoS components into an overall system suitable for the functional requirements, dependability requirements, and the characteristics of the target platform once it is known. For example, network resource reservation (which is a reusable component in our UAV prototype) is suitable only if the application requirements can make use of it (i.e., the application needs high bandwidth, low latency) and the platform can support it (i.e., the network supports bandwidth reservation).

This advanced separation of concerns enables us to create an application that is sufficiently flexible and efficient for the requirements and environment that it is targeted for. The alternative, i.e., to encode the adaptation in the functional code of the application, would result in either

1. A tremendously complex and inefficient application that covered every possible contingency in a large set of environments or
2. A more efficient, but brittle, application that is dependable only in a small set of environments.

Using middleware to knit together a unified capability. The current UAV prototype pulls together QuO adaptation, along with adaptation strategies written in the QuO toolkit, and resource reservation using RSVP. In one deployment of the UAV prototype, i.e., in the NSWC HiPer-D testbed, we integrated with a global resource manager capability. We have also begun exploring using Real-time CORBA, which is a first step towards incorporating an integrated CPU/network resource management capability as part of a complete end-to-end and scalable solution.

Experimental Results. We have installed two versions of the UAV prototype application in the Hiper-D lab at the Naval Surface Warfare Center (NSWC) and have evaluated them in the Hiper-D 2000 and 2001 series of integrated advanced concept demonstrations for the Naval Surface Ship domain. These versions have been described in earlier papers [ICDCS, UAV]. The current version described in this paper is being provided as an open-source release to researchers investigating related issues. The Hiper-D 2000 version used TCP and an earlier, less functional MPEG viewer, and illustrated the use of frame-dropping and migration of the distributor in response to excessive processor loads. The Hiper-D 2001 version uses the CORBA A/V Streaming Service, UDP, and DVDView, and combines bandwidth reservation, frame dropping, and load balancing. The current version is the one described in this paper and adds wireless networks, live camera feeds, video processing, control feedback, and contract-controlled coordination between UAVs.We ran an experiment on the second version of the UAV software, using the CORBA A/V Streaming Service over UDP, to informally evaluate the effectiveness of the adaptability strategy in focusing available resources when they are constrained. We ran a total of three runs:

1. A control run, with no adaptation
2. A second run, where adaptation is implemented by frame dropping and
3. A third run, which utilized both frame dropped and RSVP bandwidth management.

The trials were run with the sender and distributor on the same Pentium III 933 MHz processor and 512 MB of RAM, and with the receiver on a separate laptop, with a Pentium II 200 MHz processor and 144 MB of RAM, all running Linux, with a 10 Mbps link between them. We

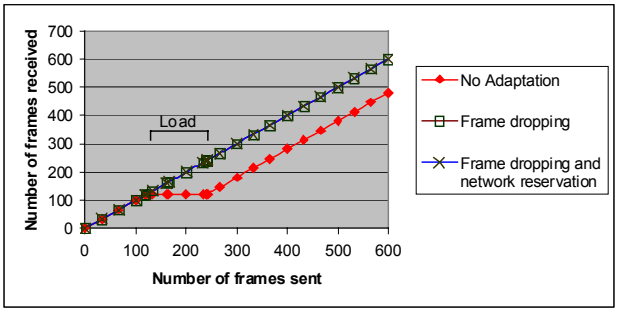

Figure 6: QuO adaptation ensured successful delivery of all video under load

started the video running. After 60 seconds, we applied a load to the network link for 60 seconds, then removed the load. After another 180 seconds, 300 seconds in all, the experiment terminated. The data collector recorded each MPEG I frame (2 per second, 600 in all) that was sent from the sender, and each that was received at the receiver, and the time elapsed from send to receive.

The results of each test run are described below:

- Test run 1, which as the control run without any adaptation, lost 119 of the 121 I frames sent while the system was under load, i.e., only 481 of the 600 I frames sent made it through. The average delay of the frames that made it through was 56.58 ms, with a median delay of 55 ms, a minimum delay of 38 ms, and a maximum delay of 121 ms. The average delay of the frames sent through when the system was not under load was 56.33 ms, with a median of 55 ms, a minimum of 38 ms, and a maximum of 67 ms. 1.65 percent of the I frames sent made it through when the system was under load (2 out of 121), with 98.35% of the I frames being lost by the UDP transport. The average delay of the two that did make it through under load was 115.5 ms.
- Test run 2, with frame dropping as its only adaptation, got all 600 of its I frames through despite the load on the system. The average delay of all the frames was 70.01 ms, with a median delay of 57 ms, a minimum delay of 50 ms, and a maximum delay of 143 ms. The average delay of the frames sent through when the system was not under load was 57.01 ms, with a median of 55 ms, a minimum of 50 ms, and a maximum of 68 ms. 100 percent of the I frames sent made it through when the system was under load (120 out of 120), with 0% of the I frames being lost by the UDP transport. The average delay of the frames delivered while the system was under load was 122.15 ms.
- Test run 3, with adaptation using both frame dropping and network reservation (RSVP), also got all 600 of its I frames through despite the load on the system. The average delay of all the frames was 64.2 ms, with a median delay of 59 ms, a minimum delay of 52 ms, and a maximum delay of 106 ms. The average delay of the frames sent through when the system was not under load was 58.1 ms, with a median of 56 ms, a minimum of 52 ms, and a maximum of 71 ms. 100 percent of the I frames sent made it through when the system was under load (120 out of 120), with 0% of the I frames being lost by the UDP transport. The average delay of the frames delivered while the system was under load was significantly lower than the other two runs, 88.5 ms.

Figure 6 illustrates the improvements afforded by the adaptation under load. The test runs that included QuO adaptation were able to recover from the load imposed on the system to keep the video flowing and not lose any important (i.e., I) frames. The video stream that did not have adaptive control lost nearly all the video sent during the time when load was imposed on the system.

3. Lessons Learned and Open Research Issues

The previous section presented two applications we have built to evaluate and validate our middleware-based concepts and ideas concerning managed QoS and adaptive behavior. We looked at these applications through a narrow lens focusing on how we use managed QoS concepts to promote dependability under non-optimal and/or changing conditions. Based on our results, we have learned the following lessons:

- Dependability needs to address working under changing requirements and unanticipated conditions
- It is feasible to operate with less than a full complement of resources, so long as they are targeted at the critical parts
- There is a context sensitive nature to “what’s the best behavior”
- Late binding is an avenue to many innovative approaches
- Layered solutions with integrated parts are an important development tool, especially for large, complex problems. This involves information sharing and cooperative behavior across and between these layers.

In this section we speculate on where these types of activities are or ought to be headed. In doing so, we take a broader view of the challenges we face in constructing the new generation of DRE systems, going well beyond the relatively simple examples developed to date. In our view, the key research challenges for the next several years will involve the integration and augmentation of the following capabilities:

- *Contracts and adaptive meta-programming* – Information must be gathered for particular applications or application families regarding user requirements, resource requirements, and system conditions. Multiple system behaviors must be made available based on what is best under the various conditions. This information provides the basis for the contracts between users and the underlying system substrate. These contracts provide not only the means to specify the degree of assurance of a certain level of service, but also provide a well-defined, high-level middleware abstraction to improve the visibility of adaptive changes in the mandated behavior. A crucial (and currently missing) capability involves the ability to change strategies rapidly and with few negative side effects.

- *Graceful degradation* – Adaptive meta-programming mechanisms must also be devised to monitor the system and enforce contracts, providing feedback loops so that application services can degrade gracefully (or augment) as conditions change, according to a prearranged contract governing that activity. The initial challenge here is to establish the idea in developers' and users' minds that multiple behaviors are both feasible and desirable. The next step is to put into place the additional middleware support–including connecting to lower level network and operating system enforcement mechanisms–necessary to provide the right behavior effectively and efficiently given current system conditions.

- *Prioritization and physical world constrained load invariant performance* – Some systems are highly correlated with physical constraints and have little flexibility in some of their requirements for computing assets, including QoS. Deviation from requirements beyond a narrowly defined error tolerance can sometimes result in catastrophic failure of the system. The challenge is in meeting these *invariants* under varying load conditions. This often means guaranteeing access to some resources, while other resources may need to be diverted to insure proper operation. Generally collections of such components will need to be resource managed from a system (aggregate) perspective in addition to a component (individual) perspective.

Although it is possible to satisfy contracts, achieve graceful degradation, and globally manage some system resources to a limited degree in a limited range of systems today, much R&D work remains. The research strategies needed to deliver these goals can be divided into the seven areas described below:

1. Individual QoS Requirements – Individual QoS deals with developing the mechanisms relating to the end-to-end QoS needs from the perspective of a single user or DRE application. The specification requirements include multiple contracts, negotiation, and domain specificity. Multiple contracts are needed to handle requirements that change over time and to associate several contracts with a single perspective, each governing a portion of an activity. Different users running the same application may have different QoS requirements emphasizing different benefits and tradeoffs, often depending on current configuration. Even the same user running the same application at different times may have different QoS requirements, *e.g.,* depending on current mode of operation and other external factors. Such dynamic behavior must be taken into account and introduced seamlessly into next-generation distributed systems.

General negotiation capabilities that offer convenient mechanisms to enter into and control a negotiated behavior (as contrasted with the service being negotiated) need to be available as COTS middleware packages. The most effective way for such negotiation-based adaptation mechanisms to become an integral part of QoS is for them to be "user friendly," *e.g.,* requiring a user or administrator to simply provide a list of preferences. This is an area that is likely to become domain-specific and even user-specific. Other challenges that must be addressed as part of delivering QoS to individual applications include:

- Translation of requests for service among and between the various entities on the distributed end-to-end path
- Managing the definition and selection of appropriate application functionality and system resource tradeoffs within a "fuzzy" environment and
- Maintaining the appropriate behavior under composability.

Translation addresses the fact that complex network-centric systems are being built in layers. At various levels in a layered architecture the user-oriented QoS must be translated into requests for other resources at a lower level. The challenge is how to accomplish this translation from user requirements to system services. A logical place to begin is at the application/middleware boundary, which closely relates to the problem of matching application resources to appropriate distributed system resources. As system resources change in significant ways, either due to anomalies or load, tradeoffs between QoS attributes (such as timeliness, precision, and accuracy) may need to be (re)evaluated to ensure an effective level of QoS, given the circumstances. Mechanisms need to be developed to identify and perform these tradeoffs at the appropriate time. Last, but certainly not least, a theory of effectively composing systems from individual components in a way that maintains application-centric end-to-end properties needs to be developed, along with efficient implementable realizations of the theory.

2. Run-time Requirements – From a system lifecycle perspective, decisions for managing QoS are made at design time, at configuration/deployment time, and/or at run-time. Of these, the run-time requirements are the most challenging since they have the shortest time scales for decision-making, and collectively we have the least experience with developing appropriate solutions. They are also the areas most closely related to advanced middleware concepts. This area of research addresses the need for run-time monitoring, feedback, and transition mechanisms to change application and system behavior, *e.g.,* through dynamic reconfiguration, orchestrating degraded behavior, or even off-line recompilation. The primary requirements here are *measurement*, *reporting*, *control*, *feedback*, and *stability*. Each of these plays a significant role in delivering end-to-end QoS, not only for an individual application, but also for an aggregate system. A key part of a run-time environment centers on a permanent and highly tunable measurement and resource status service as a common middleware service, oriented to various granularities for different time epochs and with abstractions and aggregations appropriate to its use for run-time adaptation.

In addition to providing the capabilities for enabling graceful degradation, these same underlying mechanisms also hold the promise to provide flexibility that supports a variety of possible behaviors, without changing the basic implementation structure of applications. This reflective flexibility diminishes the importance of many initial design decisions by offering late- and run-time-binding options to accommodate actual operating environments at the time of deployment, instead of only anticipated operating environments at design time. In addition, it anticipates changes in these bindings to accommodate new behavior.

3. Aggregate Requirements – This area of research deals with the system view of collecting necessary information over the set of resources across the system, and providing resource management mechanisms and policies that are aligned with the goals of the system as a whole. While middleware itself cannot manage system-level resources directly (except through interfaces provided by lower level resource management and enforcement mechanisms), it can provide the coordinating mechanisms and policies that drive the individual resource managers into domain-wide coherence. With regards to such resource management, policies need to be in place to guide the decision-making process and the mechanisms to carry out these policy decisions.

Areas of particular R&D interest include:

- *Reservations*, which allow resources to be reserved to assure certain levels of service
- *Admission control mechanisms*, which allow or reject certain users access to system resources
- *Enforcement mechanisms* with appropriate scale, granularity and performance and
- *Coordinated strategies and policies* to allocate distributed resources that optimize various properties.

Moreover, policy decisions need to be made to allow for varying levels of QoS, including whether each application receives guaranteed, best-effort, conditional, or statistical levels of service. Managing property composition is essential for delivering individual QoS for component based applications, and is of even greater concern in the aggregate case, particularly in the form of layered resource management within and across domains.

4. Integration Requirements – Integration requirements address the need to develop interfaces with key building blocks for system construction, including the OS, network management, security, and data management. Many of these areas have partial QoS solutions underway from their individual perspectives. The problem today is that these partial results must be integrated into a common interface so that users and application developers can tap into each, identify which viewpoint will be dominant under which conditions, and support the tradeoff management across boundaries to get the right mix of attributes. Currently, object-oriented tools working with distributed object computing middleware provide end-to-end syntactic interoperation, and relatively seamless linkage across the networks and subsystems. There is no *managed* QoS, however, making these tools and middleware useful only for resource rich, best-effort environments.

To meet varying requirements for integrated behavior, advanced tools and mechanisms are needed that permit requests for *different* levels of attributes with different tradeoffs governing this interoperation. The system would then either provide the requested end-to-end QoS, reconfigure to provide it, or indicate the inability to deliver that level of service, perhaps offering to support an alternative QoS, or triggering application-level adaptation. For all of this to work together properly, multiple dimensions of the QoS requests must be understood within a common framework to translate and communicate those requests and services at each relevant interface. Advanced integration middleware provides this common framework to enable the right mix of underlying capabilities.

5. Adaptivity Requirements – Many of the advanced capabilities in next-generation DRE system environments will require adaptive behavior to meet user expectations and smooth the imbalances between demands and changing environments. Adaptive behavior can be enabled through the appropriate organization and interoperation of the capabilities of the previous four areas. There are two fundamental types of adaptation required:

1. Changes beneath the applications to continue to meet the required service levels despite changes in resource availability and
2. Changes at the application level to either react to currently available levels of service or request new ones under changed circumstances.

In both instances, the system must determine if it needs to (or can) reallocate resources or change strategies to achieve the desired QoS. Applications need to be built in such a way that they can change their QoS demands as the conditions under which they operate change. Mechanisms for reconfiguration need to be put into place to implement new levels of QoS as required, mindful of both the individual and the aggregate points of view, and the conflicts that they may represent. In particular, fundamental concern in performing adaptive resource management is to reduce overhead and improve predictability of adaptation. If too much of the resource budget is spent monitoring and performing adaptive functions, therefore, adaptation can prove ineffective or even detrimental to system performance.

Part of the effort required to achieve these goals involves continuously gathering and instantaneously analyzing pertinent resource information collected as mentioned above. A complementary part is providing the algorithms and control mechanisms needed to deal with rapidly changing demands and resource availability profiles and configuring these mechanisms with varying service strategies and policies tuned for different environments. Ideally, such changes can be dynamic and flexible in handling a wide range of conditions, occur intelligently in an automated manner, and can handle complex issues arising from composition of adaptable components. Coordinating the tools and methodologies for these capabilities into an effective adaptive middleware should be a high R&D priority.

6. System Engineering Methodologies and Tools – Advanced middleware by itself will not deliver the capabilities envisioned for next-generation embedded environments. We must also advance the state of the system engineering discipline and tools that come with these advanced environments used to build complex distributed computing systems. This area of research specifically addresses the immediate need for system engineering approaches and tools to augment advanced middleware solutions. These include:

- *View-oriented or aspect-oriented programming techniques,* to support the isolation (for specialization and focus) and the composition (to mesh the isolates into a whole) of different projections or views of the properties the system must have. The ability to isolate, and subsequently integrate, the implementation of different, interacting features will be needed to support adapting to changing requirements.
- *Design time tools and models*, to assist system developers in understanding their designs, in an effort to avoid costly changes after systems are already in place (this is partially obviated by the late binding for some QoS decisions referenced earlier).
- *Interactive tuning tools*, to overcome the challenges associated with the need for individual pieces of the system to work together in a seamless manner
- *Composability tools*, to analyze resulting QoS from combining two or more individual components
- *Modeling tools for developing system performance models* as adjunct means (both online and offline) to monitor and understand resource management, in order to reduce the costs associated with trial and error
- *Debugging tools*, to address inevitable problems.

7. Reliability, Trust, Validation, and Certifiability – The dynamically changing behaviors we envision for next-generation large-scale, network-centric systems are quite different from what we currently build, use, and have gained some degrees of confidence in. In particular, since adaptive meta-programming mechanisms must monitor and respond to externally induced stimuli, they are vulnerable to malicious behavior. Considerable effort must therefore be focused on assuring the validity of inputs and administrative operations, validating the correct functioning of the adaptive behavior, and on understanding the properties of large-scale systems that try to change their behavior according to their own assessment of current conditions, before they can be deployed. But even before that, longstanding issues of adequate reliability and trust factored into our methodologies and designs using off-the-shelf components have not reached full maturity and common usage, and must therefore continue to improve. The current strategies organized around anticipation of long life cycles with minimal change and exhaustive test case analysis are clearly inadequate for next-generation dynamic systems with stringent QoS requirements.

4. Concluding Remarks

This paper began by discussing the need for adaptive behavior as a key component of making the computing and communication systems we build more responsive to the changing conditions that are often associated with distributed real-time and embedded (DRE) applications and environments. We articulated the particular viewpoint of adaptive behavior as another means to enhance dependability, by keeping systems functioning effectively under non-optimal and/or changing conditions. We examined this viewpoint in the context of two representative DRE systems that we have built using these principles, and also

in the context of related system organization issues. We then proposed an R&D agenda for the next five years, which can take these ideas and propel them into a new generation of systems, responsive both to user requirements and realities of real world environments, as well as to the software engineering challenges associated with systems of ever enlarging scope and complexity.

References

[WSOA] Corman, David, October 2001, *WSOA–Weapon Systems Open Architecture Demonstration–Using Emerging Open System Architecture Standards to Enable Innovative Techniques for Time Critical Target (TCT) Prosecutio*n, 20th Digital Avionics Systems Conference (DASC), Daytona Beach, Florida, IEEE/AIAA.

[UAV] Karr DA, Rodrigues C, Loyall JP, Schantz RE, Krishnamurthy Y, Pyarali I, Schmidt DC. Application of the QuO Quality-of-Service Framework to a Distributed Video Application. Proceedings of the International Symposium on Distributed Objects and Applications, September 18-20, 2001, Rome, Italy.

[ICDCS] Loyall JL, Gossett JM, Gill CD, Schantz RE, Zinky JA, Pal P, Shapiro R, Rodrigues C, Atighetchi M, Karr D. Comparing and Contrasting Adaptive Middleware Support in Wide-Area and Embedded Distributed Object Applications. Proceedings of 21st IEEE Intern'l Conference on Distributed Computing Systems (ICDCS-21), April, 2001, Phoenix, AZ.

[RT-ARM] Huang, Jim Jiandong; Jha, R.; Muhammad, Mustafa; Lauzac, S.; Kannikeswaran, B.; Schwan, K.; Zhao, W.; Bettati, R. RT-ARM: a real-time adaptive resource management system for distributed mission-critical applications. IEEE Workshop on Middleware for Distributed Real-time Systems and Services, 2 December 1997, San Francisco, CA.

[Kokyu] Christopher D. Gill, David L. Levine, and Douglas C. Schmidt The Design and Performance of a Real-Time CORBA Scheduling Service, Real-Time Systems: the International Journal of Time-Critical Computing Systems, special issue on Real-Time Middleware, guest editor Wei Zhao, March 2001, Vol. 20 No. 2

[TAO] D. C. Schmidt, D. L. Levine, and S. Mungee. The Design and Performance of Real-Time Object Request Brokers. *Computer Communication*s, 21(4):294–324, Apr. 1998.

[CORBA-AV:97] OMG. *Control and Management of Audio/Video Streams, OMG RFP Submission (Revised), OMG Technical Document 98-10-0*5. Object Management Group, Framingham. MA, Oct 1998.

[QuO] Loyall J, Schantz R, Zinky J, Bakken D. "Specifying and Measuring Quality of Service in Distributed Object Systems", *Proceedings of The 1st IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC 98)*, 1998.