A Unified Process for the Integration of Large-scale, Distributed, Object-Oriented, Real-time Systems in Layered Architectures

M. Mortazavi[1] and J. Connell
Teknowledge Corporation, Palo Alto, CA 94303

Abstract

Over the past few decades a great deal of research has been devoted to the development of real-time components and systems. Examples include real-time operating systems, real-time schedulers, real-time object models and real-time object brokers. Nevertheless, efforts to build such large scale systems have lagged behind due to interoperability problems, programming paradigms which are difficult to use, an absence of a standard QoS specification language, a general lack of maturity in software engineering processes involved, and logistic difficulties of building large-scale, distributed, real-time systems within reasonable expenditure of resources. This paper focuses on the initial phases of a recent effort (the QUITE project) to build a large-scale, QoS-aware, real-time system based on the integration of research technologies that have received funding from the Defense Advanced Research Project Agency's Quorum program. It emphasizes the process aspects of the QUITE integration effort.

1. Introduction

Here, we describe the initial phases of the Quorum [19] Integration, Testbed and Exploitation (QUITE) project. Funded by Defense Advanced Research Projects Agency (DARPA), QUITE has several objectives, some of which are relevant to this report. One objective has been to evolve a *component-based*, Object-Oriented architecture for a Quality of Service (QoS) aware system with end-to-end capabilities spanning timeliness, volume and reliability. Today, real-time capabilities are one of the dimensions in the QoS parameter space [25] which also include volume and reliability of service as well as data fidelity. An example of timeliness QoS parameter is the delay in data delivery, commonly known as latency. Another example is the variation in the delay with respect to time, or jitter.

A strategic goal of QUITE has been the specification of abstract objects and their interfaces. The second goal has been the adoption of multiple (enterprise, information, computational, engineering and technology) views the elaboration of architecture definition and integration. The evolving, architecture for component software developed under the Quorum program is a layered architecture as shown in Figure 1. This high-level architecture had already been implicitly proposed by DARPA [19]. Most layer names suggest their function. There are also two columns. The "Quorum Services" column stands for those functional components that provide transparency and drill down through the horizontal layers. "Distributed Objects & Object Services" collect a set of components and services that provide for a distributed objects computing paradigm based on CORBA. Several technologies are being integrated in the Distributed Objects (DO) layer to provide QoS along timeliness, volume and reliability dimensions. The real-time specifications being generated by the OMG [23] inform the DO integration but since the overall integration needs to be based on actual existing components, DO approaches which may be non-standard but currently provide timeliness QoS are included.

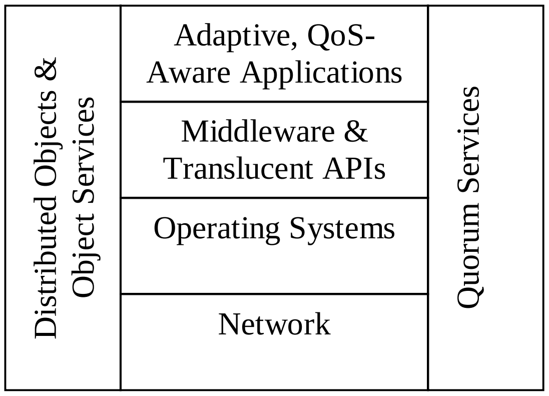

Distributed Objects & Object Services
Adaptive, QoS-Aware Applications
Middleware & Translucent APIs
Operating Systems
Network
Quorum Services

Figure 1. Integration Architecture Service and Functional Layers

Due to the fact that QoS negotiation requires awareness of QoS, available at layers that are

[1] The authors are the chief software architects and project leads for the Quorum Integration, Testbed and Exploitation (QUITE) project, funded by Defense Advanced Research Projects Agency (DARPA). The views expressed in this paper do not represent those of DARPA, Teknowledge or its contracting partners, The Open Group Research Institute and S/TDC. (Addendum: M. Mortazavi has moved to Sun Microsystems. His work on QUITE was conducted while he was at Teknowledge Corporation.)

more than one removed from any particular level, it becomes necessary to speak of the concept of *drill-down* when discussing such systems. Drill-down exposes those services of layers that are more than one removed from any given layer and are needed by the requesting layer for QoS negotiation.

Yet another goal of QUITE is the production of reference implementations for the evolving architecture. The software engineering process employed is based on incremental and iterative rapid prototyping, and we have used UML as a language for use case, class model, API, object interaction and call protocol specification. While there are alternative architecture description languages, UML remains the *de facto* standard of most software engineers and as system integrators, we are interested in providing expressive enough languages for architecture description which facilitate architectural understanding by software engineers who are engaged in implementation of the integrated system.

Quorum is a very large research program, with software components being developed at many widespread research institutions concurrently. We sought to provide an architecture definition that would unify this community around a vision of the system to be implemented as a result of the integration of their software. More rigorous software description languages such as Z may provide a greater level of conciseness in software specification [18] and architecture. We selected UML for QUITE because it provided the right balance to make it useful to people with a range of backgrounds in formal languages, including managers and software developers. We have found UML to be highly useful in this respect. It is the specification language most likely to be understood by any random member of the software engineering community. UML is also in widespread enough use to justify a need for acquiring the literacy where it is lacking. UML was also considered rich enough to capture the broad as well as the specific constraints imposed by requirements. It provided the QUITE architects not only with a means to reflect the requirements onto the growing software specification but also with a means to conduct design activities in a uniformly understood language. Finally, the Object Management Group (OMG), a respected and industry-backed standards body, has selected UML as its object design language.

Our rapid prototyping approach when applied to a multi-site, multi-component, cutting-edge technology such as Quorum includes a WWW-augmented development environment and software configuration management, tele-video conference which bring great savings in coordination of effort, centralized requirements commissioning, continuous monitoring, testing and dynamic validation of components.

This report covers the activities carried out for the first phase of this three-year project and summarizes future plans. The following sections describe our architecture definition process, research component solicitation process, component evaluation process, our rapid prototyping approach and the current components being integrated into the first prototype of our reference implementation (with a brief justification for the selection of these components) and our future plans.

2. The Component Integration Process

In proposing this high-level integration architecture, our multi-site, multi-organization integration team of software professionals sought to maximize our ability to identify research components, to evaluate their growing levels of maturity and to apportion them to the right layer or service within an easily comprehensible high-level, layered architecture. Very early on, we identified eight major categories of functional capabilities need to be present in order to meet our requirements: real-time OS extensions, resource management, adaptive applications, real-time distributed objects and object model technologies, group communications and fault tolerance, dynamic validation and a test harness.

2.1 Architecture Definition

The architecture definition has been carried out in continuously emerging and evolving detail, in keeping with requirements set by our target customers as specified by DARPA, and in conformance with our growing understanding of available research and COTS technologies. Specifically, we used the Unified Modeling Language (UML) [1, 7, and 8] to describe software packages, class composition of packages, object interfaces, use cases, interaction and deployment diagrams. It should be noted that while there has been much work to extend UML in order to accommodate real-time design and development, no standard methodology exists for designing real-time OO systems. Therefore, architects and designers have used their own modes of extension (using UML's inherent extension possibilities) to design, model and reason about real-time behaviors of systems. A particular example is Douglass's work as presented in [6]. However, OMG's Real-time

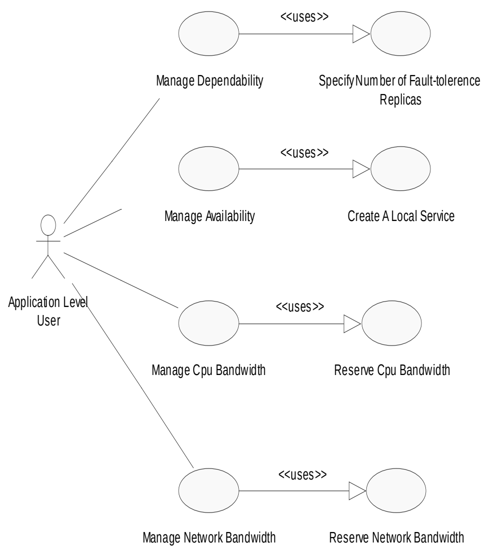


Figure 2. A Partial View of A Use Case For Resource Management Through Reservation

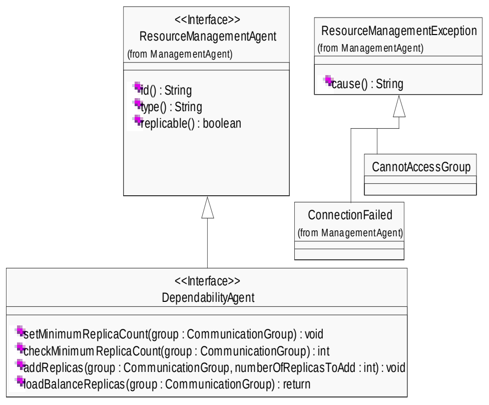
Analysis and Design working group is developing requirements to be included in an RFP (request for proposal) to be released for real-time extensions to UML. Principle concerns here are the inclusion of time semantics.

In our rapid prototyping approach, we do not attempt to force the requirements definition to completion and hurry to develop a complete detailed design specification before beginning to implement the prototype system. In fact, the prototype is used to discover, iteratively, the true and complete system requirements, specification and design. We specify an increment, implement an increment and repeat in small such increments until we have both a specification and a system that represents the true set of complete requirements. This process is well documented in a text on object-oriented rapid prototyping [3].

We have found that the tools available within UML provide ways to model all necessary system characteristics. In fact, there is a growing danger that UML, once a relatively simple notational convention with a very few model types, is experiencing growth that will tend to make design completion overly cumbersome and brittle for modern systems. Modern systems tend to have severe time and budget constraints. This is so even for complex domains such as Quorum. There seems to be a tendency for sets of notational conventions used in specification methodologies to become increasingly obese (some would say robust) over time, adding new notations and new models to capture yet more of the characteristics of the systems to be designed. This is understandably natural and yet it may lead to the notational convention eventually becoming too cumbersome for general use without significant tailoring. In our opinion, UML may already have arrived at this point in its evolution.

We are required on QUITE, for instance, to have a complete reference implementation prototype the first year and a complete, ready-for-use, end-to-end Quality of Service system within three years. We cannot afford to spend a year on design. We have, therefore, selected a subset of available UML modeling tools wherein we feel that 20% of the tools will provide 80% of a complete design specification and, further, that 80% will be sufficient to implement a high quality system within time and budget constraints.

As an example, Figure 2 provides a partial view of one of the simple use cases defined for Resource Management (RM) which is among a large set of QoS services under the umbrella of Quorum Services. Figure 3 provides some object interfaces that we have found representative of a large number of resource managers developed within the Quorum research program. In this particular case, we have specified those interfaces within Resource Management services implemented to provide management of some pre-existing dependability or fault-tolerance service based on object replicas. In other words, these are Resource



<<Interface>>
ResourceManagementAgent
(from ManagementAgent)
id() : String
type() : String
replicable() : boolean
ResourceManagementException
(from ManagementAgent)
cause() : String
CannotAccessGroup
ConnectionFailed
(from ManagementAgent)
<<Interface>>
DependabilityAgent
setMinimumReplicaCount(group : CommunicationGroup) : void
checkMinimumReplicaCount(group : CommunicationGroup) : int
addReplicas(group : CommunicationGroup, numberOfReplicasToAdd : int) : void
loadBalanceReplicas(group : CommunicationGroup) : return

Figure 3. Examples of Preliminary Interfaces to Support Dependability Management within a QoS Management System

Management interfaces for arranging the appropriate provision of replicas for fault-tolerance. No research-based resource management system within Quorum that we presently know of provides this kind of dependability management in a QoS-aware systems. AQuA, which is a replica management system developed at the University of Illinois, Urbana-Champaigne, contains a starting point in the direction of implementing such interfaces [4]. (Currently, standard fault tolerance services are being specified by the OMG [22].) Part of the problem lies in the fact that some research groups have to go beyond their areas of expertise to provide for needed functionality and interfaces which other research groups are more capable and expert in designing and implementing.

Using other UML diagrams such as interaction diagrams, we can express negotiation among entities within the functional architecture layers. Examples of these figures are not shown here but our whole architecture can be viewed at our web site [21]. We are using Rational Rose as our CASE tool but we have built additional tools to provide HTML versions of the UML models.

There are inevitable discrepancies, such as the one briefly described above, between existing research software at various levels of maturity and the evolving Quorum architecture. Nevertheless, we believe our interaction with the research groups developing the Quorum research software will eventually close all major gaps. We post regular monthly updates to our architecture specification so that we may collect appropriate feedback from various stakeholders and target customers affected by our system integration work.

2.2 Component-based Architecture and Rapid Prototyping

Our integration approach is based on long experience gained by the QUITE project management in rapid prototyping techniques [3]. It is also informed by previous, positive and negative experience gained in the design and development of integrated systems. For example, our team's experience indicated that integration projects such as the Distributed Computing Environment (DCE [11]) which were driven by existing components, rather than by a desire for architectural purity, were generally very successful. On the other hand, other projects, *e.g.*, "Distributed Management Environment" or DME[2], which was primarily driven by an emphasis on architectural purity in the absence of existing components in critical functional domains, had much less success. The QUITE team opted for the first approach, *i.e.*, the component-based architecture approach, which places emphasis on existing components for rapid prototyping of a system based on an architectural specification that provides guidance and rigorous specification for component integration.

2.3 Component Solicitation and Evaluation Process

The QUITE project seeks to meet the needs of specified technology transfer targets (external non-DARPA customers). This goal is met by working with the Quorum research community to incorporate existing technology into a single system that will meet those needs. There may be more than one reference implementation for that system. DARPA's Quorum research program has already developed a number of approaches to various aspects of the QoS problem. A key goal of the QUITE project, which has also been the subject of the present report, is to develop an architectural framework, along the lines summarized above, that can adopt the most relevant existing Quorum research components. We have specified, through formal architectural description methods based on UML, a level of detail that will enable us to incorporate selected components into an integrated system based on our architecture definition.

As a way to solicit components from the researchers, we used a Component Solicitation Form (CSF), whose content was modeled after those recommended in the request for technology (RFT) letters used in the DCE integration effort by The Open Group, in order to identify and characterize the technologies that may be integrated into the reference implementations in as seamless a manner as possible. The CSF is not a formal specification document. While some level of uniformity was expected in the set of questions answered by a CSF, the format and level of specificity of a CSF submission was expected to vary depending on the technology under consideration and the technical goals of the Quorum reference implementation.

[2] Our presentation of the case for DME are based on private communications with OSF architects such as David Black. DME never got off the ground.

More than 40 components were submitted from 28 research organizations. The submissions were in the areas of adaptive applications, global and local resource management, middleware, distributed objects, dependability components and object paradigms, object models and real-time object execution engines, network QoS components for both implicit and explicit active networks, as well as backwards-compatible packet flow and video frame compression protocols. The evaluation criteria for measuring the degree of fitness for integration of submitted components was derived using a fair, open, and consensus-based process, that is intended to provide sound procedures of selecting components for integration into a working whole.

The primary purpose of the CSF was to solicit contributions from the Quorum and other QoS related research communities of technologies to be used in the Quorum integrated reference implementation. The CSF helps to ensure that all reasonable alternatives are evaluated, not just those known to the integrator team. The CSF was also useful for sampling the existing state of a technology, according to the claims of the technology developers, in order for the integrator team to make appropriate decisions about independent construction, purchase or postponement of particular component functionality.

It is expected that a new CSF will be issued for each reference implementation in order to acquire those components, which add the capabilities required by each one of the yearly reference implementations over the next three years. Future implementations will be built on top of existing systems to the extent possible. Because of this, compatibility with the current baseline reference implementation will be an expected evaluation criterion in future solicitations.

The CSF is generated by the evaluation team formed by the Integrator. The evaluation may include experts from the Quorum research community from time to time to provide both critical knowledge and a balanced evaluation. The evaluation team will act as the focal point for screening of responses. The CSF is posted on the QUITE Web Site where researchers will find an easy web-based automated process for sending the integration team their CSF along with requested contact information. It may also be mailed to vendors and universities who are known to be working in the target technology area.

In the first component solicitation cycle which was completed in October of 1998, the evaluation team evaluated the submitted responses based on collaboratively established criteria (with input from the Quorum research community). We prioritized the responses according to weights based on the evaluation criteria. We expect these criteria to evolve as we approach the final integrated whole. We feel that this provides a ranking that favors components having the best chance of providing the required integrated capabilities in the project time frame.

Members of the evaluation team then visited the suppliers of highly ranked submissions to make a final determination. We could afford this selection process because many more components were selected than we have time or resources to integrate in the first year of the project. Selected components will be acquired and incorporated into the Reference Implementations. The final component selections will be announced to the Quorum community.

It is important to note that selection and evaluation is not intended as a judgment of the merits of any single research project. Instead, it is intended to meet the specific needs of the current Quorum integration project, QUITE. There will be additional evaluation and selection phases to meet evolving Quorum needs as determined by the technology transfer targets and the Quorum architecture.

2.4 Component Selection and Feature Injection for Integration and Testing

Component Selection. As described earlier, our process for component selection consisted of three phases: *(i)* self-selection, *(ii)* evaluation based on claims, and *(iii)* on-site evaluation of component development teams. In the first phase, we requested the researchers to fill out and submit to us a Component Solicitation Form (CSF) using a web-based process. Those who did not submit a CSF could be considered to have self selected out of our near term integration.

The CSF phase gave us 40 components to select from. We then formed a six-person component selection team and came up with a set of component evaluation criteria that could be derived from a CSF submission and used to score each of the 40 submissions. Generally speaking, we initially evaluated components on the basis of four attributes:

1. Near term availability,

2. Ease of integration within the planned technology base,
3. Extent component satisfies specified domain requirements,
4. Extent to which component is implemented for general use.

These attributes were evaluated by ascertaining from the CSF whether 1) the source code exists and will be delivered in a form that can be compiled and executed in our QUITE testbed, 2) the component was developed in an object-oriented, standards-compliant manner, 3) the component fits nicely into one of our architecture packages and therefore fulfills specified requirements, 4) the component was written using good software engineering practices so that it will be robust, scalable, stable, and maintainable.

We developed a scoring matrix that contained these criteria and the weights the selection team assigned to them. The team then put scores for each component into the matrix and the weighted sums of the scores were used to select a smaller list for site visits. During site visits, we were shown demonstrations, had an opportunity to ask detailed questions about each component and learned about the kind of environment needed to host each component. We were then able to determine whether the developers will be available to provide support for the integration effort

We learned, from an examination of acquired components, that a fairly complete preliminary reference implementation could be assembled for our preliminary demonstration using a minimum of one component in each of our six primary architecture layers, shown in Figure 1. The list of some of the components selected for integration in our first reference implementation during the first phase include TMO [10, 15], AquA [4], EPIQ [5], CEDAR [16], MSHN [24], DeSiDeRaTa [27], TAO [14], HPF [26], Ensemble [9], NetSimQ [17], QUASAR components [25], and Darwin [2]. A detailed description of each of these research projects, including other selected components, can be found at the DARPA-sponsored Quorum web site [19].

In the first implementation, which will include some of these components, we are seeking to test interoperability and to incorporate several QoS features including soft real-time. The components were selected according to the criteria discussed earlier in this paper. In our initial demo application, we are focusing on demonstration of component capabilities in the areas of group communication and replica management, resource management, and distributed objects, maximizing the QoS capabilities that can be demonstrated in an integrated system that gains QoS characteristics in an incremental fashion, optimizing the inter-play of an architectural vision and available software component building blocks.

In general, guaranteed QoS solutions for a distributed, dynamically evolving system with process migration and replication should be required to provide metrics on variations given feasibility conditions imposed by an analysis of system requirements and available resources. As system integrators for DARPA we have been asked not only to rely on our requirements commissioners and their system engineers to have performed some feasibility studies but also to assist them. However, our major concern is to provide an engineering approach such that once requirements are met with the best system specifications, we can guarantee minimum violation of these requirements within a dynamical system that may be subject to fault and recovery.

Feature Injection. In integrating components, it is always easiest to select one computing model and build a whole system according to that computing model. However, in an integration effort, reference implementation of a system architecture must demonstrate the possibility of interoperability among computing paradigms used by components which are best suited to provide a particular service within a layer or a particular QoS feature across layers. QoS features have been injected into layers within our distributed demonstration and test application. Each QoS feature, real-time ones, as well as non-time dependent ones such as fault-tolerance, are examined within that particular architectural layer where the feature is implemented, while at the same time the interoperability of components are put to a hard acceptance test. Figures 4 and 5 show the layers within our distributed demo application and motivate the idea of *feature injection* into these layers for test and integration purposes.

2.5 Demonstration Application as Focus for Rapid Prototyping

One of the criteria of success in most integration projects is the demonstration of capabilities obtained. In our rapid prototyping approach [3], we have selected possible demonstration applications and use cases to drive our efforts in architecture definition.

The group of reference applications need to include multiple mechanisms involved in application models (e.g. path/DAG-based distributed computing, streaming, etc.), application/object distribution transparencies (e.g. in access, location, failure, migration, relocation, replication, persistence and transaction) and communication technologies (e.g. asynchronous or synchronous message passing, remote procedure calls, object invocation, etc.). While building towards such broad objectives, constructing gradually evolving demonstration application early-on has several advantages and benefits:

1. improving familiarity with research as well as COTS technologies and components,
2. iterative definition of the layered architecture in the light of that familiarity, and
3. incremental development of real-time and QoS features within various layers of the application.

As one concrete example, the application layers in our Command and Control (C2) reference application model, used to implement our initial system demonstration scenario, consist of:

1. A layer for a *simulated world* with the greatest possible stringency on real-time requirements. Currently, this is implemented using TMO [10, 15] components from a TMO test application called CAMIN, with some simple modifications that enable them to run on a single Pentium II, 400 MHz machine.

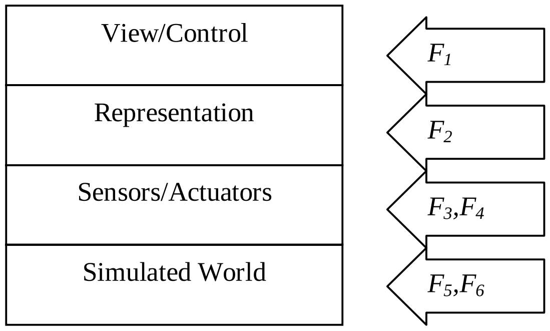


Figure 4. Feature Injection into Distributed, Application Layers

2. A layer of *smart sensors* and sensing *engines* that detect messages from the simulated world and parcel them into events to be dispatched to the next layer up. Currently, TAO is used as the ORB in this event-based communication. TAO's QoS-aware event channel is being employed to provide differentiated treatment of high-value as opposed to subsidiary, low-value events. TMO object can be used here to provide timed dispatch of sensing data. In a C2 application timeliness requirements usually become "softer" above this layer, meaning that guarantees are required and given on less strict range of variations.

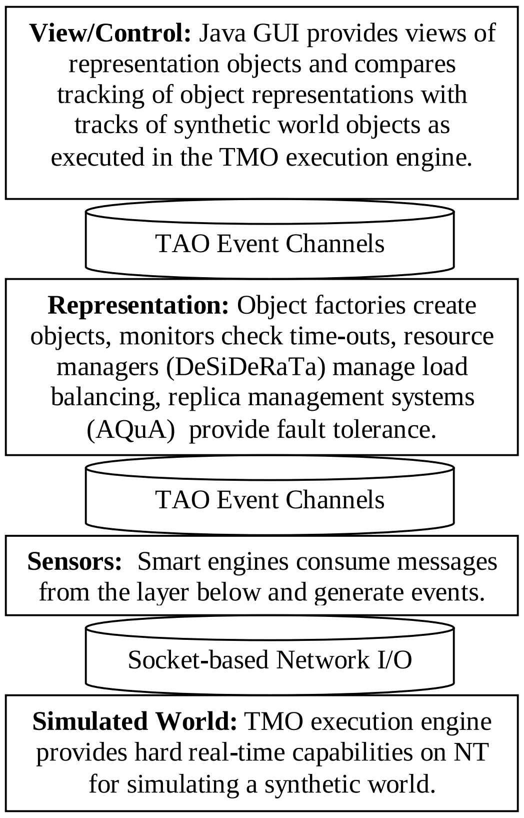

View/Control: Java GUI provides views of representation objects and compares tracking of object representations with tracks of synthetic world objects as executed in the TMO execution engine.
TAO Event Channels
Representation: Object factories create objects, monitors check time-outs, resource managers (DeSiDeRaTa) manage load balancing, replica management systems (AQuA) provide fault tolerance.
TAO Event Channels
Sensors: Smart engines consume messages from the layer below and generate events.
Socket-based Network I/O
Simulated World: TMO execution engine provides hard real-time capabilities on NT for simulating a synthetic world.

Figure 5. An Example of Reference Applications

3. A layer of *representation objects* representing the objects in the simulated world. The representation objects, which also interpreted raw events from the senor/event engines, are instantiated using object factories (the CORBA Lifecycle Service, to be more precise). They are timed-out if no new messages are sent for their updates. Since regular house-keeping is necessary for these objects much can be gained from the incorporation of the TMO model for object execution. Object factories

are distributed and take account of objects they have created. A data structure is used to persist the state of objects, whether they are created, ready to be destroyed, or tagged to continue receiving events. Currently, we are using our own simple implementation of object factories. In the future, we will use a full-scale, standard CORBA Life Cycle service.

4. A layer of *view/control objects* that are used in the GUI application to view the representation objects. Events are published from the representation objects, and the view objects subscribe to these events. As of this writing, we have not implemented the control and actuation objects. These will wait for an implementation of harder real-time requirements that are due to come in the second and third year of QUITE.

The demo application will consist not only of the C2 viewer, but will also contain a distribuetd system viewing and QoS control capabilities. Currently, we have specified means for viewing the status of a network of processor nodes. Finally, at this stage, each layer of the demo application will have a different set of QoS and real-time capabilities as discussed in the feature injection subsection above.

In summary, the synthetic/simulated world layer will have relatively hard real-time capabilities gained by the use of TMO. The layer-to-layer communication will have some level of QoS due to the differentiated treatment of events gained by use of TAO's CORBA event service. The representation layer will have fault-tolerant capabilities where the removal of one of the representation replicas will leave others behind to take over in a consistent manner. Except for the last characteristic (fault-tolerance in the representation layer), we have been successful in integrating the other features, *i.e.*, the hard-real-time feature of TMO and the QoS-aware event channel from TAO. However, we have not yet added fault-tolerance to the representation layer. An important concern in any integration effort involving QoS is the demonstration of the interoperability of component software that can provide various types of QoS.

3. Issues

Although we have focused on matters of process, we will include some technical issues in this section. We have had to face some major problems in this early phase of our integration effort.

The first problem arises from the diversity of QoS specification languages and a lack of agreement on a universal specification language that will meet all researchers and technologists needs. Our experience with existing languages show that a language such as XML can be ideal for specification of real-time requirements for a hierarchical, distributed (i.e. multi-node, multi-system), heterogeneous (in hardware, operating systems and network gear) application subdivided into computationally distinct tasks. Resource management at all levels and in all components need to use a component-independent language for QoS specification, one that allows for easy creation of sub-dialects without the need for large variations in semantics, a need for meta-translators, new reference volumes and users guides and varying compiler technologies. XML provides such a solution.

The second problem, for a dynamically evolving system, is proper deployment of monitoring needed in order to reapportion resources to migrating objects and processes. However, the proper tradeoffs between monitoring and using resources can prove hard to achieve. High priority processes can defeat the purposes of high resolution, low priority monitoring. High priority, low resolution monitoring can defeat the purpose of optimal resource allocation.

Actual research and technology components come on multiple platforms. While we have achieved a great deal of integration using Windows NT-based systems, much relevant components have been developed on Linux. We are dealing with the problems by deploying a heterogeneous environment. The ultimate goal for the project has become the provision of a heterogeneous platform for distributed, QoS computing.

Our interaction with the research and technology community and our own experience in building distributed computing environments have shown that applications with distinct computational tasks can be organized as directed acyclic graphs. The nodes represent the tasks and the arches the data paths. The abstraction can be a powerful model that covers many application domains, including multi-media and C2. Working application models/paradigms are critical for productivity in software engineering. Limiting the application models to DAG-based applications can provide a lowest possible denominator for component integration and standardization.

Finally, we have found out that requirements are often hard to gather and many are kept secret for national security reasons. In such circumstances, system complexity should be kept at a minimum, application models should be standardized, component integration modularized and integration approach made versatile and robust to respond to changing requirements.

4. Guarantees in a Dynamical System

We end this report with a short discussion of how one might profitably define guarantees within a highly dynamical system suffering continuous faults, recovery, process/object migration, network failures and congestion, plugable, mobile devices, intermittent, unpredictable requests for services. Guarantees for "best possible services" can always be achieved (and defined) by an over-provisioning of computational and communication resources.

In Dynamical systems composed of DAG-based applications running on distributed resources under duress of the sort described in the previous paragraphs, guarantees are required on variations about an expected mean, on variations in response time to recover from failures, etc. We will consider the first of these as an example.

The task of the system designer is to minimize sources of such variation by judicious deployment of engineering solutions.

For the purposes of illustration consider a hypothetical system where all functions are continuous in time to the extent needed. Let us focus on the overall delay on a chain, Δ_o, in a DAG-based system $\Delta_o = \Sigma\ \Delta_i$, where Δ_i is the delay caused in the performance of the i-th task and the communication delays caused as the data is moved from the i-th task to the next task along the chain. The overal delay Δ_o is equal to $\Sigma\ \Delta_i$, the sum of individual delays. In a dynamical system where the members of the chain, the composition of the chain itself and the physical distribution of the chain can vary with time, the individual delay can be modeled (in rudimentary terms) as follows: $\Delta_i = \Delta_i\ (D(c(t)),t)$, where D is the physical distribution of the chain, and t is the time. This time-dependent evolution of the delay represents the dynamics in our simple dynamical system. The variation in delay at the i-th stage can be obtained by taking the time derivative of the equation above to get $\partial_t \Delta_i + \partial_D \Delta_i.\ \partial_c D.\ \partial_t c$. To reduce variation (about a mean) of the overall, one must reduce the variation of each task involved in the chain. But to achieve this, one must select tasks such that as they are performed by an object on a node, the performance will not vary highly with time, that the variation of physical distribution of objects with changes within the chain (due to failure or splitting to replica objects performing the same task) should be kept low, so should the dependence of the delay of a particular task on the changes in the physical distribution of that particular task, and finally faults and recoveries should be minimized. Those are pretty simple ideas. Their implementation has been puzzling technologists, researchers and system-integrators alike for some time, indicating that simple ideas can sometimes be prohibitively difficult to conceive and implement.

5. Conclusion

QUITE is a QoS and real-time system integration project. In this project, we have been able to identify a list of potential research components for integration within a single layered functional architecture as well as a separate, layered application object architecture. We have learned that while the research components are not equally mature, they have usually been very well designed for interoperability. We have been able, in our preliminary effort to integrate Quorum research components, to build a multi-layer C2 application, where layer-to-layer communications among all but 2 layers is through a QoS-aware CORBA event channel.

We have shown that it is effective to implement an integrated system that involves heterogeneous research components by using a software engineering practice that involves:

- a careful component solicitation process,
- a distributed layered demo application,
- a feature injection scheme and
- an architecture specification methodology.

Acknowledgement

We would like to acknowledge the collaboration and assistance of Amir Bahmanyari, Jim Davidson, Slava Galperin, David Lounsbury, Art Robinson and Slava Vaynman in the architecture, component solicitation and system integration work discussed here. We would also like to thank Alan Sandlin (SPAWAR) and Gary Koob (DARPA) who have supported our work.

References

1. Alhir, S. S., *UML in a Nutshell*, O'Reilly & Associates, 1998.
2. Chandra, P., Fisher, A., Kosak, C., Ng, T., Steenkiste, P., Takahashi, E., and Zhang, H. "Darwin: Customizable Resource Management for Value-Added Network Services" *Sixth IEEE International Conference on Network Protocols* (ICNP'98), Austin, October 1998
3. Connell J. and Shafer L., *Object-Oriented Rapid Prototyping*, Yourdon Press Computing Series, 1994.
4. M. Cukier, J. Ren, C. Sabnis, D. Henke, J. Pistole, W. H. Sanders, D. E. Bakken, M. E. Berman, D. A. Karr, and R. E. Schantz, "AQuA: An Adaptive Architecture That Provides Dependable Distributed Objects" *Proceedings of the 17th IEEE Symposium on Reliable Distributed Systems (SRDS'98)*, West Lafayette, Indiana, USA, October 20-23, 1998.
5. Z. Deng, J. W.-S. Liu, L. Zhang, M. Seri, and A. Frei. An open environment for real-time applications. (To appear in *Real-Time Systems Journal*, 1998.)
6. Douglass, B. P. *Real-Time UML: Developing Efficient Objects for Embedded Systems*, Addison-Wesley, 1998.
7. Eriksson, H. and Penker, M., *UML Toolkit, John Wiley & Sons*, 1997.
8. Harmon, P. and Watson, M., *Understanding UML*, Morgan Kaufman Publishers, 1997.
9. Hayden, M., *The Ensemble System*, Cornell University Technical Report, TR98-1662, January 1998.
10. Kim, K.H. (Kane), "Object Structures for Real-Time Systems and Simulators" *IEEE Computer*, August 1997, pp.62-70.
11. Open Software Foundation, *Introduction to OSF DCE : revision 1.0*, Prentice Hall, 1992.
12. *Real-time Analysis and Design Working Group*, Object Management Group Conference, November 9-13, Burlingame, California, USA.
13. Sabnism, B. S., *Proteus: A Software Infrastructure Providing Dependability for CORBA Applications*, (98SAB01.pdf, 220 KB) Master's Thesis, University of Illinois, 1998.
14. Schmidt, D., Levine, D., and Mungee, S., "The Design of the TAO Real-Time Object Request Broker" *Computer Communications*, Special Issue on Building Quality of Service into Distributed Systems, Elsevier Science, Volume 21, No. 4, April, 1998.
15. E. Shokri, P. Crane, K. Kim, "An Implementation Model for Time-Triggered Message-Triggered Object Support Mechanisms in CORBA-Compliant COTS Platform" *Proceedings for the First International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'98)*, Kyoto, Japan, April, 1998.
16. Sivakumar, S., Sinha, P., and Bharghavan, V. "Core Extraction Distributed Ad Hoc Routing (CEDAR) Specification", Internet draft. (draft-ietf-manet-cedar-spec-00.txt)
17. Tyan, H., Wang, B., Ye, Y. and Hou, C., "NetSimQ: a Java-integrated network simulation tool for QoS control in point-to-point high speed networks," presented in the Work-in-Progress session of *IEEE Real-Time Technology and Applications Symposium*, Denver, CO, June 1998.
18. Woodcock, J. and Davies, J., *Using Z: Specification, Refinement, and Proof*, Prentice Hall International Series in Computer Science, 1996.
19. http://www.darpa.mil/ito/research/quorum/index.htm
20. http://www.crhc.uiuc.edu/PERFORM/AQuA.html
21. http://quite.teknowledge.com/architecture/topview.html
22. http://www.omg.org/techprocess/meetings/schedule/Fault_Tolerance_RFP.htm
23. http://www.omg.org/homepages/realtime/
24. http://www.mshn.org/
25. http://www.cse.ogi.edu/DISC/projects/quasar/
26. http://www.timely.crhc.uiuc.edu/HPF/
27. http://desidrta.uta.edu/~project/
28. http://www.cs.uoregon.edu/research/qos/