Interactors: Capturing QoS and Resource Requirements Between Multiple Cooperating Objects

Richard West and Karsten Schwan

College of Computing
Georgia Institute of Technology
Atlanta, GA 30332

Abstract

Complex distributed applications, including virtual environments, and real-time multimedia require quality of service (QoS) guarantees in the end-to-end transfer of information across a network. For many applications, these QoS requirements are formulated on application-level objects and are dynamic, since they may change during the execution of the application. To support dynamic QoS guarantees requires the management of processing, memory, and network resources. Consequently, we introduce 'interactors', which capture the dynamic QoS and resource requirements at each stage in the generation, processing, and transfer of information between multiple cooperating objects.

1 Introduction

Complex distributed applications, including virtual environments, and real-time multimedia require quality of service (QoS) guarantees in the end-to-end transfer of information across a network. QoS requirements for many distributed applications, such as groupware applications, virtual environments (e.g. DIVE), and distributed interactive simulations(DIS), are formulated on application-level objects. Typically, these requirements are dynamic; they may change from one object invocation to the next, they may change at intervals determined by simulation speed, by reaction rates of human participants, or by the rates of change in input behavior. For example, to avoid collision, a virtual flight navigation system might require one user piloting a plane to be updated frequently with complete, detailed (loss-free) state information about other planes in the nearby air-space. When two plane entities are sufficiently far apart, however, their state-updates with respect to position, orientation and graphical detail may not need to be exchanged as frequently. Furthermore, as two objects move apart in a virtual world, each user controlling one of these objects might only require approximate information about the other object. Similarly, when viewing continuous sensors in this virtual world, certain sensors might be important at some particular time (e.g., a radar sensor when targeting an enemy plane) whereas other sensors might be irrelevant (e.g., a GPS location indicator during enemy engagement). Moreover, sensor visibility changes over time, depending on pilot viewpoint, for example.

In the above example, each user must maintain cached (or shadow) copies of the state information about entities manipulated by other users, operating on remote hosts in a distributed system. For these types of shared objects, there is a linkage between information consistency and QoS requirements: (1) a suitable consistency protocol must be used to maintain the correct level of consistency between replicated state information, and (2) timing or quality-of-service (QoS) requirements must be placed on the update and access to distributed shared objects whose state has been partially replicated.

Our work is based on the following assumptions: (1) dynamic resource management is necessary or, at minimum, important to attain desired end-to-end service quality and corresponding levels of resource usage, and (2) a strong linkage exists between the temporal and/or spatial consistency of information exchanged between objects, and the quality of service applied to such information exchanges. Since both object interactions and service requirements vary at runtime, abstractions beyond those of static service contracts, costly admission control and policing are needed.

This paper describes one such runtime abstraction, called an 'interactor'[1], to capture the dynamic QoS and resource requirements at each stage in the generation, processing, and transfer of information between multiple cooperating objects. An interactor is an association between cooperating shared objects spanning a network and all of the resources, both communication and computation bound, that support information exchanges between objects. Furthermore, interactors are a convenient vehicle for deciding at which level of abstraction QoS requirements can and should be translated; they capture the actual and required levels of service and resource usage, so that this information can be be used for the on-line adjustment of service quality and requirements across multiple levels of system abstraction.

To demonstrate the use of interactors, we are building a system to coordinate both the thread and packet-level scheduling, necessary to maintain service quality when information is exchanged between application-level objects. Information exchanged between objects is carried in invocations, which are ultimately scheduled and then marshalled into messages. At the network-level, messages are fragmented into packets and scheduled for transmission with packets carrying information for other object interactions.

Our work provides the necessary abstractions to maintain QoS constraints in synchrony with each application's execution. Furthermore, using interactors, we can control interactions between all forms of resources required by, and services provided for, cooperating objects.

In the next section, we describe the object-level characteristics and, hence, the QoS requirements for the distributed real-time applications targeted by our work.

2 QoS Requirements for Objects

Complex distributed real-time applications, such as virtual environments, have the following object-level characteristics:

- *Dynamic exchanges between objects* - information exchanged between objects interacting across a network is generated dynamically, as a result of changes in the working environment of the application. For example, information is exchanged between cooperating objects as a result of two entities moving within range of each other in a virtual environment application.
- *Variable rates of exchanges between objects* - object exchanges are often bursty. Namely, there are periods of time when very few exchanges take place, whereas at other times, many exchanges may occur between objects. Factors affecting exchange rates include the proximity of entities in virtual environments. For example, more frequent updates about the location of such entities may be necessary as entities move closer together.
- *Variable resolution of exchanges between objects* - the information exchanged between objects may vary in detail, depending on the importance of detail at the time information is exchanged, for instance. If two entities in a virtual environment or DIS application are semantically 'closer', it might be necessary to increase the resolution, and therefore bandwidth, of information exchanged between the objects corresponding to each entity.
- *Variable resource demands due to adaptable objects* - two or more distributed objects involved in the exchange and processing of information might have several methods for the same objective. For example, a GPS system used for calculating the positions of entities in a 3D virtual space might execute one of several possible algorithms; the chosen algorithm depends on the required accuracy of the entity's position, and the increased computation and memory costs of executing a more 'precise' algorithm.

QoS requirements for (and constraints on) information exchanged between objects in applications like those described earlier include:

- *Delay* – this constraint specifies the total tolerable end-to-end delay on information exchanged between objects, before the information is no longer valuable.
- *Loss-tolerance/fidelity* – at the object-level, this specifies the desired level of detail of information that must be exchanged. Fidelity constraints bound the degree of loss or detail of information exchanged between objects.
- *Throughput* – this is necessary to specify the rate of exchange of information between objects. Throughput is bounded by the maximum and minimum frequency of updates to shared objects of known granularity (size).
- *Consistency constraints* – these constraints bound the degree of coherence between shared objects. Strict coherence of objects limits scalability in terms of the total number of objects that can be replicated, since more object exchanges are required as the number of replicas increases. By exploiting application-level temporal and spatial constraints on shared objects, greater responsiveness and scalability can be achieved, compared to running the same application on a system that does not exploit such constraints. The actual format of these constraints is in terms of a *semantic function* [2], which specifies the conditions about *when* and *with which objects* updates should occur. The specification of these semantic-based consistency requirements can only be made in the context of the application. For virtual environment applications, we can impose not only 'temporal' constraints (which specify *when* updates should be applied) but also 'spatial' constraints on consistency. Spatial consistency constraints ensure that updates are applied to a copy of a shared object only when that object is within a certain range of another process's (user's) object. For example, one user ($p1$) manipulating one object might not care to know about another user's ($p2$) object but $p2$ might want to know about $p1's$ object in shared space because it is within a range of interest to $p2$.

Having presented the fundamental characteristics and QoS requirements for objects in the distributed real-time applications targeted by our work, we now describe QoS and resource management using interactors.

3 QoS and Resource Management Using Interactors

Existing systems supporting distributed applications lack the necessary mechanisms to allocate resources to meet the service requirements of application-level objects. Many researchers (e.g., in the Tenet group) have focused on service guarantees at the network-level , but there is still the issue of providing service guarantees in the end-systems, meaning a separate agent must be responsible for this task. The problem with this approach is how to integrate the agents responsible for guaranteeing service in the network and the end-systems.

We propose the ‘interactor’ as a key abstraction in supporting dynamic QoS and resource management amongst cooperating, distributed, application-level objects. Interactors provide a general mechanism for providing QoS for multiple applications, thus alleviating the system implementer (or application developer) of creating ad-hoc solutions for specific applications.

Paths and Interactors: As stated earlier, an *interactor* is an association between cooperating shared objects spanning a network and all relevant resources, both communication- and computation-bound, that support object exchanges. Interactors capture the QoS and resource requirements at each stage in the generation, processing, and transfer of information between multiple cooperating objects. Interactors extend the notion of a ‘path’ (as used in the x-kernel for establishing configurable communication protocols) by capturing associations between multiple interacting objects and their QoS and resource requirements. Namely, a path is a point-to-point logical connection, along which messages flow through a set of statically-defined processing modules and associated resources. Interactors have similarities to *connections* in QuO but they also influence where in a path system adaptations should be applied to achieve a given level of service. For example, an interactor for a video server might decide to compress video frames, or drop packets at the packet scheduling level, to reduce demands on link bandwidth.

An example of interactors is given in Figure 1. Two processes access different consistency protocol modules in the consistency management layer but use the same error control, fragmentation and packet scheduling modules in the network layer of the sending host.

Interactors in Adaptive Systems: QoS requirements are specified by the application and vary depending on the nature of the application. (NOTE: We refer to the application-layer as the *end-point* for an interactor.) The

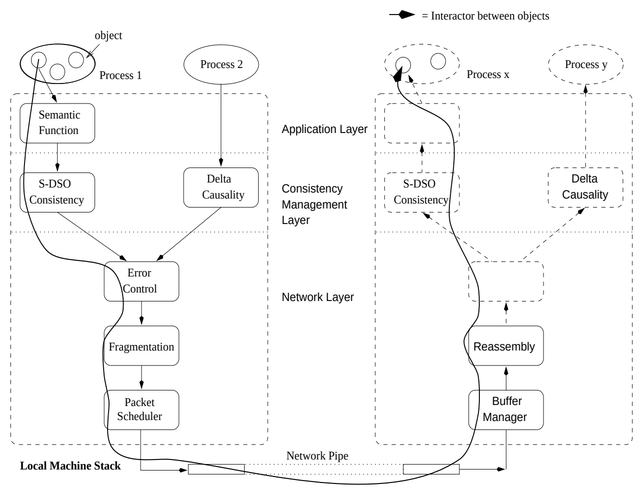

Figure 1: An example showing interactors between objects in different processes. Interactors are the end-to-end associations between cooperating objects and all resources along the path between those objects. Interactors also capture the QoS requirements at all stages of information flow between objects.

problem with specifying quality constraints at an end-point is ensuring the specification is simple yet descriptive enough to ensure the supporting adaptive system has enough information to allocate resources most appropriately. Furthermore, the system must completely understand at what level of abstraction, quality requirements can be translated into resource (or service) constraints.

For example, consider a video capture application, which requires the generation and transfer of video frames from a server to multiple clients. Each client can tolerate 50% loss of video frames. Loss-tolerance is specified as a QoS constraint at the client end-points. An adaptive system supporting this video application can apply these quality constraints to determine corresponding service and, hence, resource constraints at one of several stages in the interaction path. The loss-tolerance may be translated at the network-level to ensure that an error-control module retransmits or forward-error-corrects video frames to guarantee each client receives a minimum of 50% of all frames. Alternatively, a different interactor module may duplicate video frames, thereby consuming more network bandwidth due to the transmission of redundant information. If the loss constraints are applied at the packet scheduling level, the scheduler could decide to select only half of all frames, thereby reducing network-bandwidth demands. Finally, the loss-tolerance may be translated into CPU constraints at the (computation) thread-level. In the latter case, the frames might be encoded using, for example, a lossy-compression algorithm. Observe that an error-control module employing forward-error-correction would also place demands on CPU resources.

The above example illustrates that end-point quality requirements place different demands on resources depending on the point in the interaction path at which they get translated. Moreover, if there is a choice of several hierarchical stages at which QoS requirements can be translated, the adaptive system must know at which stages it is eligible to perform translation. To do this, the application end-point specifies an interactor by supplying the system not only a description of its QoS requirements but also a *signature* defining the abstract module path of an interactor. The semantics of end-point QoS requirements at various stages of an interactor depend on the corresponding module at each stage, and the service already received by an interactor at the current stage in its path. In fact, an interactor's signature defines exactly where and to what extent QoS requirements are translated.

3.1 Dionisys: Runtime Support for End-to-End QoS

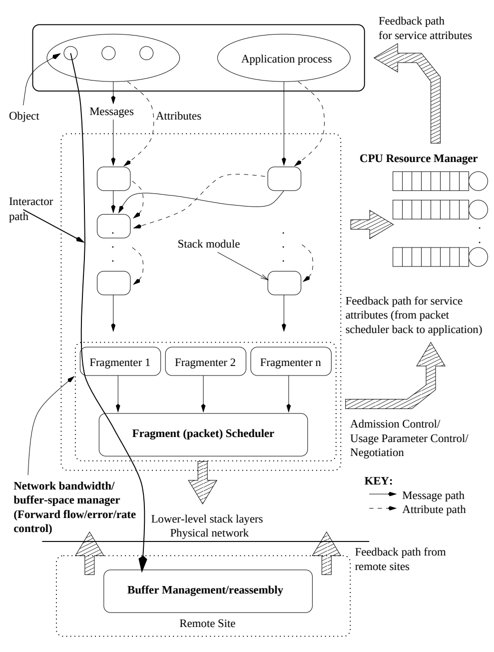

Figure 2: Dionisys: Runtime Support for End-to-End QoS

We are developing a prototype runtime system, called Dionisys (see Figure 2), to support end-to-end QoS in distributed real-time applications, such as virtual environments with real-time state-sharing constraints.

Application-level objects specify interactors, which include QoS requirements (specified as service attributes) and processing signatures, determining at which stages of an interactor path QoS translations can be applied. For each module in a path, the constraints are mapped to a form meaningful to that module.

In Dionisys, the CPU resource manager is responsible for binding modules to threads (or processes) and scheduling threads on processors. The network manager is responsible for fragmenting and reassembling messages, packet scheduling, and policing/flow/error control. Service attributes may be specified for an interactor when the interactor is first established, or they may be dynamically set by a message transmitted across an interactor. Interactors can either be established before messages are exchanged between objects, or when message-exchanges begin. The latter approach would be most applicable for short, infrequent message-exchanges. In the extreme case, a message traveling across an interactor may specify its own unique service attributes, allowing each individual message traveling along the same interactor to specify its own QoS requirements. If admission control/negotiation is in effect, the network manager feeds back to the application layer the optimum level of service it can provide, given the desired service level the application requires and the maximum service the system can supply in the presence of other messages requiring service.

4 Conclusions and Future Work

This paper introduces 'interactors', which capture the dynamic QoS and resource requirements at each stage in the generation, processing, and transfer of information between multiple cooperating objects, in real-time distributed applications such as virtual environments. To demonstrate the use of interactors, we are currently developing a prototype runtime system, called Dionisys.

We are still working on the infrastructure to specify interactors, so that dynamic QoS specification and translation of requirements on information exchanged between cooperating objects can be supported.

Future work includes developing thread and packet co-scheduling algorithms, so that both communication and computation resources can be coordinated to maintain service qualities on information exchanged between application-level objects.

References

[1] R. West and K. Schwan, "Interactors: Capturing tradeoffs in bandwidth versus cpu usage for quality of service constrained objects," Tech. Rep. GIT-CC-98-03, Georgia Institute of Technology, January 1998.

[2] R. West, K. Schwan, I. Tacic, and M. Ahamad, "Exploiting temporal and spatial constraints on distributed shared objects," in *Proc. 17th IEEE ICDCS*, IEEE, 1997.