

# Dynamic Application Structuring on Heterogeneous, Distributed Systems<sup>1</sup>

Saurav Chatterjee

SRI International, Menlo Park, Ca 94025

*saurav@erg.sri.com*

<http://www.erg.sri.com/projects/erdos>

**Abstract.** The diversity of computers and networks within a distributed system makes these systems highly heterogeneous. System heterogeneity complicates the design of static applications that must meet quality-of-service (QoS) requirements. As part of the ERDoS<sup>2</sup> project, we have introduced a novel concept of dynamic application structuring where the system, at run time, chooses the best end-to-end implementation of an application, based on the system and resource attributes. Not only does this approach improve resource utilization and increase the total benefit to the user over that provided by the current static approaches, but it also is transparent to users and simplifies application development. We describe the models and mechanisms necessary for dynamic application structuring and use a set of multimedia applications to illustrate dynamic application structuring.

## 1 Introduction

Systems are becoming more and more heterogeneous. As the general use of faster and faster desktop computers increases, so does that of the relatively slower laptops and handheld machines. Similarly, the increased use of faster and faster networks (such as ATM<sup>3</sup> and gigabit Ethernets) is accompanied by the increased use of the much slower wireless networks and modems.

Designing quality-of-service (QoS)-based applications for this type of heterogeneous environment is difficult. Static applications that attempt to provide the same level of QoS irrespective of the attributes of the underlying system, are unsatisfactory because applications whose design assumes high-performance resource attributes are guaranteed to miss QoS on lower-performance resources. Applications whose design assumes a lowest common denominator can provide only minimal levels of QoS to everyone. It is impractical to design multiple versions of an application, each tailored for a particular system, because such applications can be used only by sophisticated users who have enough expertise to pick the best version. Second, the constant introduction of new and better resources requires continuous upgrades to such an application, which in turn requires a major development effort.

The SRI ERDoS<sup>4</sup> project has developed an innovative approach to this problem— at run time, the middleware dynamically creates customized applications and keep user's QoS requirements and the system and resource attributes. We have developed

---

<sup>1</sup> The work discussed in this paper is funded by DARPA/ITO under the Quorum program, Contract N66001-97-C-8525, and was performed by SRI International (SRI).

<sup>2</sup> ERDoS: End-to-End Resource Management of Distributed Systems.

<sup>3</sup> ATM: Asynchronous Transfer Mode.

<sup>4</sup> ERDoS: End-to-End Resource Management of Distributed Systems.

an application model that captures multiple ways to structure an end-to-end application; at run time, based on the current system and resource state, the ERDoS middleware chooses the structure that best meets the user's QoS requirements. This process is transparent to the user, enabling these applications to be run by almost anyone. ERDoS also dynamically creates distributed applications on the fly at run time, so that the development effort to create distributed applications is also minimized.

Our approach increases the probability of achieving a high level of user QoS in highly heterogeneous systems. Whereas a static application may fail because a particular resource is overloaded, our dynamic approach can find alternative application structures that use other resources but still provide the same end-to-end functionality and QoS. Furthermore, static approaches may reject an application if all system resources are highly utilized. Instead, our dynamic approach can utilize alternative application structures that provide partial QoS (which is better than no QoS).

For example, a handheld device may not have the computing power to decompress MPEG<sup>5</sup> video in real time. A static application that uses MPEG compression will not be able to run on this device; a sophisticated user may, at best, be able to request JPEG<sup>6</sup> compression instead of MPEG. ERDoS can take an even more sophisticated approach, one that is completely transparent to the user. ERDoS can autonomously utilize a proxy machine close to the handheld device to convert MPEG to JPEG, so that the data between the video source and the proxy can be MPEG compressed, while the data between the proxy and handheld device is JPEG compressed. Thus, ERDoS achieves the same level of QoS as a sophisticated user switching from MPEG to JPEG, but also achieves better network resource utilization, thereby enabling other applications to potentially achieve higher levels of QoS.

Another example illustrates graceful QoS degradation via dynamic application structuring. Suppose audio-video data is sent from a source to several users, one of whom is connected via a slow modem line. Attempting to transfer both audio and video over the modem line guarantees that timing requirements will be missed. Instead, ERDoS can dynamically choose an application structure that will filter out the video and transmit only the audio over the modem line.

The remainder of this paper describes dynamic application structuring in ERDoS. Section 2 provides our definition of QoS. Section 3 describes our models to capture the QoS behavior and requirements of applications and the system. Section 4 describes how our architecture aids in dynamic application structuring. In Sections 5 and 6 we describe how the applications are developed, instantiated, and executed within the ERDoS middleware. We summarize our work in Section 7.

---

<sup>5</sup> MPEG: Motion Picture Experts Group.

<sup>6</sup> JPEG: Joint Photographic Experts Group.

## 2 Definition of QoS

We define QoS via six orthogonal dimensions (*orthogonal* in the sense that a set of these dimensions can be used to capture QoS requirements). The six dimensions are shown in Figure 1. For any data, the QoS dimensions capture the following

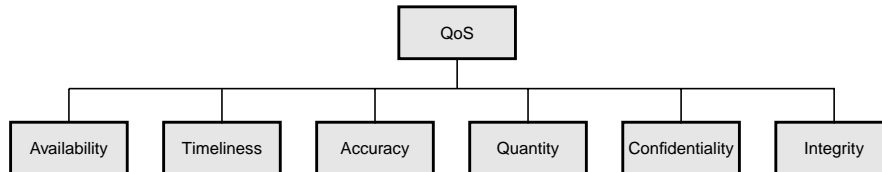


Fig. 1. ERDoS QoS Dimensions

information: what is the data (the *quantity* and *accuracy* QoS dimensions); when must it be available (*timeliness*); who has read (*confidentiality*) and write access (*integrity*); and during what percentage of time, within a time window, must these requirements be met (*availability*).

For example, a video application may demand a frame size of  $640 \times 480$  pixels (quantity); a frame rate of 25 frames/s (timeliness); and jitter of 30 ms (timeliness).

## 3 Modeling a Distributed System

The determination of the best application structure requires an understanding of the trade-offs between different application structures, their resource usage, and the QoS they can achieve. This information is captured by means of our application, resource, and system models and benefit functions. The application model captures the multiple ways to structure an end-to-end application. The benefit function captures the relative importance of the different QoS parameters, from the user's perspective. The resource and system models capture system and resource attributes. Each is described below in detail.

### 3.1 Modeling Resources

Each Resource Model (RM) captures the QoS attributes and scheduling behavior of a single computing, communication, or storage resource [1]. We recently augmented our RM to include security attributes: specifically, how long it would take, in CPU cycles, for an intruder to break the encryption scheme or to modify secure data. For example, these attributes can be used to capture the security characteristics of a virtual private network (VPN). A VPN that provides DES encryption at the network level can specify confidentiality strength on the order of  $2^{56}$  or  $2^{168}$  for triple-DES encryption.

The System Model (SM) captures the set of resources constituting the system, the topological and administrative structure of the system, and policies and objectives governing each of these administrative domains [1]. Each system is modeled as a set of subsystems, similar to Internet domains. Each subsystem also has security metrics that specify the overall confidentiality and integrity strength within that subsystem.

### 3.2 Modeling Applications

ERDoS uses the Logical Application Stream Model (LASM) [1] to store the structure of an end-to-end application. At its simplest, a LASM is represented as a directed graph whose nodes correspond to Units of Work (UoWs) and whose edges dictate the order in which the UoWs execute. A UoW is an atomic module of work, on a single resource, that transforms the application's QoS. This decomposition of an application into UoWs that transform data and QoS is a fundamental concept of our application model. UoWs are the smallest components of applications that are used for allocation and routing. The LASM captures application structure at a *logical*, or system- and user-QoS-independent level.

Figure 2 illustrates a simple video-playback application modeled via the LASM. The boxes correspond to UoWs. Because the LASM is logical, there is no need to

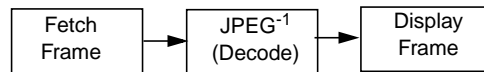


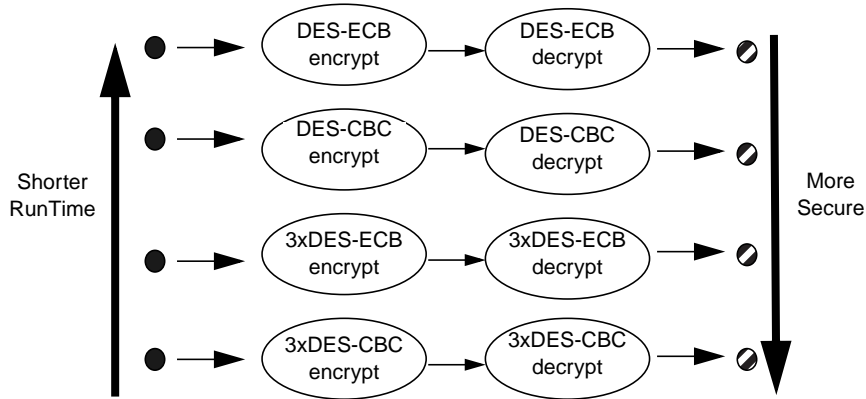
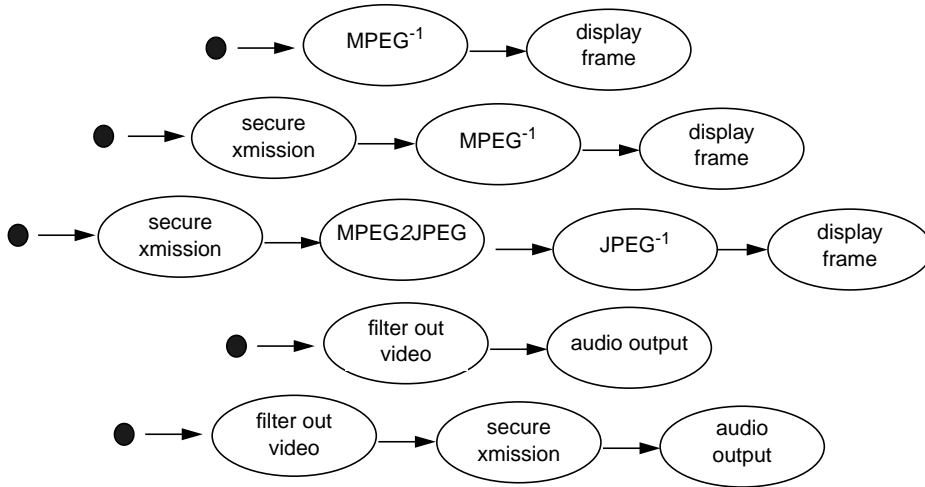
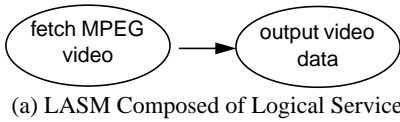
Fig. 2. Video LASM

explicitly identify I/O resources. This simple LASM facilitates portability and enables reallocation and rescheduling as the system state changes. We next describe an improvement upon this model, a recursive LASM that additionally enables ERDoS to dynamically structure applications. The recursive model is also ideal for modeling large-scale applications.

The structure of a recursive LASM is also represented as a directed graph, but the graph nodes now correspond to Logical Services (LSs). Graph edges now dictate the order in which the LSs execute. Each LS is realized by means of a Logical Realization of Service (LROs), which is composed of either a set of child LS, executing in a specific order, or a single LUoW. LUoWs, as before, are not decomposable. The structure of an LROs is identical to that of an LASM, in that it is represented as a directed graph, whose nodes also correspond to LSs. This model is recursive in that an LS is realized by an LROs, which in turn is composed of a set of child LS, which in turn may each be realized by an LROs. The recursion terminates when an LS is realized by a LUoW.

An example, a recursive video-playback LASM, is shown in Figure 3. In contrast to the example in Figure 2, the application developer does not need to specify a particular type of video decompression method at the top layer (Figure 3a); instead, the developer indicates the need for a "OutputVideoData" logical service.

Assume, for this example, that ERDoS finds five LROs for "OutputVideoData" (Figure 3b). The first three use video compression to compress the video; the last two filter out the video and only transmit audio. These last two are best when network bandwidth constraints make video transmission impractical. The first and the fourth provide no security, while the others do. The first is preferable within a secure intranet; the third is needed when the client is a simple appliance computer without enough computing power to decompress MPEG in real time; and the second is preferable under most other circumstances. With the third LROs, ERDoS can utilize a proxy machine close to the client to convert MPEG to JPEG, so that the client can then decompress JPEG format in real time.



- Input Data Symbol
- ◐ Output Data Symbol
- Logical Service

**Fig. 3. Recursive LASM for Video**

Each "secure xmission" LS is furthermore realized by four LRoSs (Figure 3c). In this example, we use DES and triple-DES to encrypt and decrypt data. According to Schneier [2], these algorithms can be implemented with either CBC or ECB blocks. (CBC is approximately 15% slower than ECB but is more secure.)

This example illustrates how the LASM provides ERDoS the flexibility to structure applications in accordance with system state. Thus, the LASM specifies different ways to structure an application and ERDoS chooses the best combination at run time, based on a variety of factors including end-to-end QoS requirements and the state of the system.

Each UoW has multiple attributes [1], but a key attribute is its Resource Demand Model (RDM). The RDM specifies the resources required (e.g., CPU cycles, network bandwidth) to achieve different levels of QoS for the current UoW. This attribute is a key factor in determining which LRoSs to use.

### **3.3 Benefit Function**

Each application can be structured in multiple ways. While many structures will achieve the same end-to-end QoS, some will not. We use benefit functions to specify relative importance to the user of the different QoS parameters for the application. ERDoS then uses this information to decide which structure will maximize QoS under the given resource constraints.

The benefit function is a  $n$ -dimensional function, where  $n$  is the number of QoS metrics of relevance to a specific application. The benefit function encodes the benefit the user receives as a function of QoS along the  $n$  dimensions.

The benefit function for an audio-video application typically has at least eight QoS metrics: frame jitter, frame size, frame rate, image clarity, audio quality, confidentiality, and integrity. While the QoS parameters are application specific, benefit is application content specific.

For example, consider two streaming video applications, one for a movie and another for a video conference between two staff members of a company discussing highly proprietary information. Clearly, confidentiality is paramount for the latter but not important for the former. On the other hand, a switch from audio-video to audio only minimally reduces benefit for the video conference, whereas the switch would greatly reduce benefit for the movie. This and other relative-importance relationships are captured using the benefit functions.

## **4 ERDoS Architecture**

ERDoS does not store prelinked distributed applications. Instead, it dynamically creates distributed applications at instantiation time. It requires only that application developers create single-resource UoWs, which are strung together to create each distributed application. This process is completely autonomous and is transparent to users. In this section, we describe how the ERDoS architecture lends itself to the process.

ERDoS is a distributed resource management middleware composed of Resource Agents (RAs), Application Agents (AAs) and a System Manager (SM), as shown in Figure 4. Most resource management decisions are made at the SM, which is thus the central component of ERDoS. The SM also

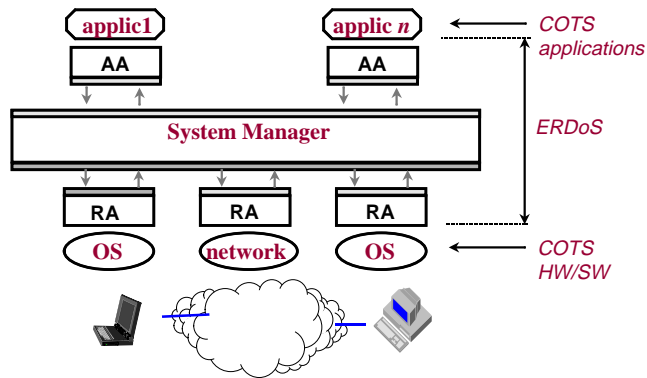


Fig. 4. ERDoS QoS Architecture

acts as a coordinator between the activities of the local resource agents and applications. The RAs (one per resource) and AAs (one per application) carry out the SM's commands and send it status updates. Each will be described in the following subsections.

Each AA is a wrapper around the application; it adds QoS monitoring and adaptation capabilities. The SM is a central entity that tracks the global picture of the system resources and applications. It controls resources via the RAs. A centralized system manager is inherently not scalable. We address this problem by using a hierarchical architecture for a distributed SM [3].

#### 4.1 System Manager

The SM uses our adaptive resource management algorithms to efficiently manage the shared resources in the distributed system. Given end-to-end application specifications, the SM determines which resources to allocate to each application [4] and how to schedule all applications on shared resources [5]. It then conveys this information to the resource agents. The SM also degrades application QoS when needed. It determines which applications to degrade and along which QoS dimensions [6], and conveys this information to the application agents and the resource agents.

The SM coordinates the activities of all the applications in the distributed systems. It starts by deciding if an application can be admitted to the system, then instantiates the different UoWs that constitute the application at the different resources. The run time coordination between the different UoWs is also the SM's responsibility.

In large systems, a system can decompose into a set of subsystems, and the ERDoS SM can be decomposed into a set of subsystem managers. Each subsystem manager then manages all the applications that only utilize resources within that subsystem. Applications that span across multiple subsystems are handled by higher-level (parent) subsystem managers.

## 4.2 Resource Agents

The SM manages the shared resources of the distributed system by coordinating the activities of the local resource agents. RAs execute the SM's commands, monitor the QoS of each work module (the UoWs) on that resource, and notify the SM whenever a UoW misses its QoS. This architecture allows the RA to be resource specific, and shields the SM from the peculiarities of the heterogeneous resources.

Each RA coordinates the different UoWs executing on a resource and provides the control interface for the UoWs. When a new application arrives at the system, the SM instructs the RAs to instantiate a new application and a set of UoWs. All future control commands to the UoW from the SM are communicated via the RA. The RA provides the infrastructure for the inter-UoW communications, and the communication between the RA and the UoW. The UoW does not directly communicate with the SM. Also, the RAs do not communicate with each other. All coordination between RAs is managed via the system manager.<sup>7</sup>

Data flow between UoWs on the same resource are handled via a RAM disk. From a UoW's perspective, RAM disks behave like normal disks and export the standard file API: i.e., UoWs access them via standard methods available in the java.io classes such as *readLine()* or C/C++ methods such as *read()*. However, instead of storing the content on a physical disk, RAM disks store their content in memory, enabling very quick reading and writing. Our RAM disk code is embedded inside the RA. We chose RAM disks for I/O between UoWs because most off-the-shelf software expect files for input and output. Using RAM disks, these software modules can be made into UoWs without expensive and time-consuming reimplementations.

Data flow between UoWs on different machines is slightly more complex than the communication within a single machine. We handle inter-resource data flow by introducing two UoWs, one on the source side that sends data via TCP (or RTP over UDP) and the second on the receiver side that receives data. These UoWs, not visible at the application user level, are dynamically added by RAs as needed.

## 4.3 Application Agents

The SM coordinates the end-to-end application via the AAs. AAs translate application-specific, end-to-end QoS requirements into common, end-to-end QoS metrics [7] that ERDoS understands. For example, a video application's application-specific QoS parameters such as video frame size and clarity are translated into common volume and accuracy QoS parameters by the video application's agent. The use of AAs enables new applications, with application-specific QoS parameters, to execute on ERDoS without the need to modify the ERDoS SM.

An AA's functionality is distributed over each UoW that the application comprises. We implement these functionalities as application wrappers, where each application wrapper is a Java class or a C library. The individual UoW wrapper interfaces with the RAs. The AAs also allow the SM to dynamically reconfigure the application as the resource status changes.

---

<sup>7</sup> The majority of data traffic is between UoWs; therefore, the SM sets up individual data channels between these UoWs. Inter-RA and RA-SM communication is relatively rare in our architecture; hence, a central SM does not constitute a bottleneck.

## 5 Application Development and Instantiation in ERDoS

In this section, we describe the ERDoS development environment and our GUI-based tools, which easily store UoWs and specify LASMs, LRoSes, RMs, and SMs.

### 5.1 Application Developer's Environment

UoWs are Java or C/C++ methods that are written by programmers. To create a distributed application, the programmer must also specify all the UoWs that constitute a distributed application and must provide communication channels to transmit data between UoWs. A key limitation is that the programmer must a priori identify the resources on which each UoW will reside. Many distributed applications, such as LBL's *vic* [8], sidestep this problem by asking the application *user* to run the source and sink modules separately and to then specify the IP address and port number. This approach assumes that the user is sophisticated enough to supply this information and, even if this assumption is correct, is practical only when applications are simple (e.g., two UoWs on two machines). This approach clearly is not scalable when a complex, distributed application contains UoWs running on many machines.

The ERDoS development environment solves these problems. It requires programmers to create UoWs only by using standard C/C++ or Java development tools and to link with our application wrappers. The programmer then specifies the application structure in a *logical*, or system-independent, manner, using the System Development Tool (SDT) GUI. The SDT is based on our earlier System Engineering Workbench (SEW) [9], which was developed while the author was at Carnegie Mellon University. The ERDoS run-time environment, described in Section 6, then determines the structure of the application and the resource on which each UoW will execute, and creates communication channels to convey data between UoWs. This process is transparent to application users, so that almost everyone can use it. It is also transparent to application developers, significantly reducing development time.

The SDT has three GUI components—applications, resources and the system, and benefit functions. The GUI enables users to specify how *a-priori*-created UoWs are strung together to create end-to-end applications. The GUI enables *nonprogrammers* to create these end-to-end applications, because ERDoS does the actual interfacing between UoWs; the user has only to specify the structure of the application and the end-to-end QoS requirements of interest to that application. These applications are then placed in the Application Repository. This GUI can also be used to enter benefit function information, but that capability is not shown. The Resource Development GUI similarly is used to capture system state, including topology and resource attributes.

### 5.2 ERDoS Instantiation Environment

The application developer creates the component UoWs of the application and the LASM describing the application structure. This information is stored in the application repository for use by the users. The Instantiation GUI enables a user who wishes to run an application to choose it from the Application Repository. The user can also choose the content to be associated with the application. The SM then indexes into the Application Repository, finds the LASM for that application, and dynamically

strings together UoWs to create an end-to-end application. QoS requirements are found from the BF associated with the content chosen by the user. The exact set of UoWs used to create the application is chosen by the application structuring algorithm. The SM then allocates resources to this application and schedules the application on these resources.

## 6 ERDoS Run-Time Environment

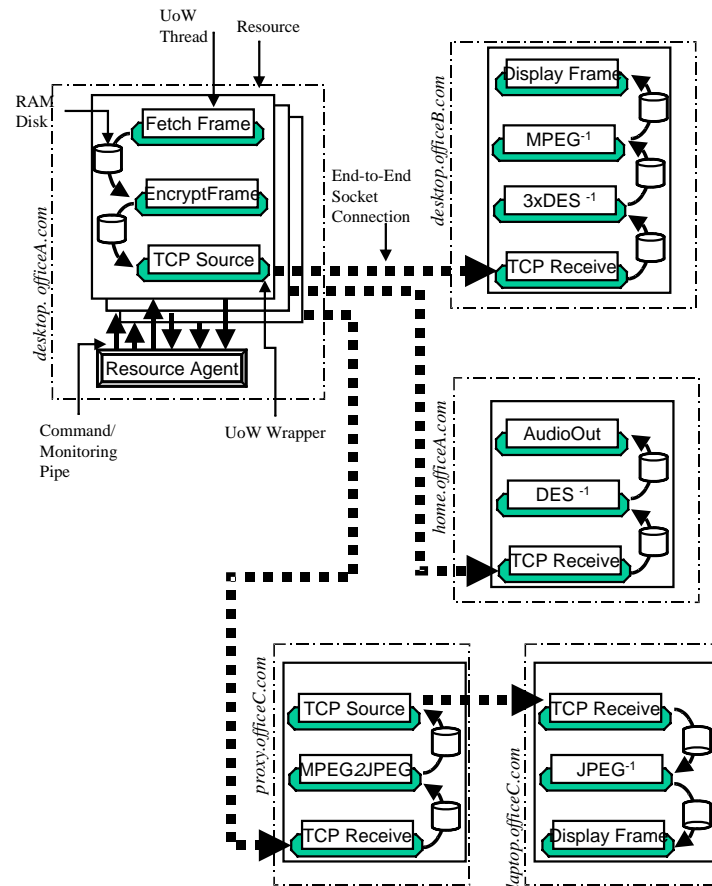
The ERDoS SM provides full resource management functionality, including allocation and scheduling, adaptation, and application structuring. After an application has been requested by the user, the SM instantiates it by invoking the appropriate UoWs at the different RAs. During run time, the SM coordinates the activities of the RAs and monitors the state of the system and the applications via the monitors inside the RAs. It also adapts to the changing system state and instructs the RAs to take corrective actions when faults occur. In the following discussion we describe the functions performed by the ERDoS system during the application's run time.

Figure 5 shows the ERDoS run-time infrastructure when a person in office A (at *desktop.officeA.com*) instantiates a audio-video conference with three other participants. The person at office B (at *desktop.officeB.com*) is connected over the public Internet. The person at *home.officeA.com* is connected over a slow modem directly to his or her office machine. The person at *laptop.officeC.com* is connected over a virtual private network (VPN). All machines except the laptop are powerful desktop machines.

Based on the resource constraints and the application structure, the ERDoS SM determines that the network connection between *desktop.officeA.com* and *desktop.officeB.com* is insecure, and decides to use the LRoS that provides strong triple-DES-CBC encryption. Consequently, it instantiates seven UoWs: "fetch frame," "3xDES-CBC encrypt frame," "transmit frame over network via TCP," "receive frame over network via TCP," "3xDES-CBC decrypt frame," "decompress frame," and "display frame." Using this application structure, the SM instructs the appropriate RAs to instantiate the seven UoWs. The first three UoWs are instantiated by the RA on the source side ("fetch frame," "3xDES-CBC encrypt frame," and "transmit frame over network via TCP") and the remaining four by the RA on the destination side ("receive frame over network via TCP," "3xDES-CBC decrypt frame," "decompress frame," and "display frame").

The SM also determines that the network bandwidth between *desktop.officeA.com* and *home.officeA.com* is highly limited, and that timeliness-related QoS parameters will not be met if both audio and video are transmitted over the modem line. It also determines that the phone is moderately secure. Therefore, it chooses the appropriate LRoS that filters out video and then encrypts the audio. This procedure results in seven UoWs: "fetch frame," "filter out video," "DES-CBC encrypt audio," "transmit data over network via TCP," "receive data over network via TCP," "DES-CBC decrypt audio," and "output audio." For this application, the RAs instantiate four UoWs on the source side and three on the destination side.

Because the office A and office C are connected over a highly secure VPN, no additional encryption is necessary for the last application. However, the SM



**Fig. 5.** ERDoS Infrastructure for Five Resources (Two Video Source Resources, Two Video Sink Resources, and a Proxy Resource) [The <sup>-1</sup> superscript denotes a decompression or decryption UoW]. The System Manager is not shown. There also exists a RA on each machine, but only the RA on desktop.officeA.com is shown.

determines that laptop.officeC.com is not fast enough to decompress MPEG and still meet all timeliness parameters. To solve this problem, the SM uses a proxy machine that translates MPEG to JPEG video. This transformation enables the laptop user to view the video.

This dynamic structuring of an application, based on resource constraints, is a direct consequence of the modular and extensible architecture of ERDoS. The last application is now represented by a LASM consisting of eight UoWs. The SM instructs the appropriate RAs to instantiate eight UoWs. The first two are instantiated by the RAs on the source side (“fetch frame” and “transmit frame over network via TCP”). The next three (“receive frame over network via TCP,” “MPEG to JPEG,” and “transmit frame over network via TCP”) are instantiated by the RA on the proxy machine (*proxy.org143.office.com*). The final three UoWs are instantiated by the RA on the destination side (“receive frame over network via TCP,” “decompress frame,”

and “display frame”). The structure of each application is chosen by our allocation algorithm [4], which bases its decision on the available LROs, the system state, and the resource attributes.

## 7 Conclusion

Distributed systems are becoming increasingly heterogeneous. Developing a static application that can work in heterogeneous systems and still meet QoS requirements is increasingly difficult. In ERDoS, we introduce an innovative method to alleviate this problem by dynamically structuring applications at run time, based on system and resource attributes. Our approach increases the probability of achieving a high level of application QoS in highly heterogeneous systems. Whereas a static approach may fail because a particular resource becomes overloaded, our dynamic approach can find alternative application structures that use other resources but still provide the same end-to-end functionality and QoS. Furthermore, static approaches may reject an application if all system resources are highly utilized. Instead, our dynamic approach can utilize alternative application structures that provide partial QoS (which is better than no QoS at all).

## References

1. Chatterjee, S., Sabata, B., Sydir, J., and Lawrence, T.: Modeling Applications for Adaptive QoS-based Resource Management. *Proc. 2nd IEEE High-Assurance System Engineering Workshop* (1997)
2. Schneier, B.: Applied Cryptography, Protocols, Algorithms, and Source Code in C. 2nd ed. John Wiley and Sons, Inc. (1996)
3. Sabata, B., Chatterjee, S., Sydir, J., and Lawrence, T.: Hierarchical Modeling of Systems for QoS-based Distributed Resource Management. Unpublished draft. SRI International, Menlo Park, California (1997)
4. Chatterjee, S.: A Quality of Service based Allocation and Routing Algorithm for Distributed, Heterogeneous Real Time Systems. *Proc. 17th IEEE Intl. Conf. Distributed Computing Systems (ICDCS97)*, Baltimore, Maryland (1997)
5. Chatterjee, S., and Strosnider, J.: Distributed Pipeline Scheduling: A Framework for Distributed, Heterogeneous Real-Time System Design. *Computer Journal* (British Computer Society), Vol 38(4) (1995)
6. Sabata, B., Chatterjee, S. and Sydir, J.: Dynamic Adaptation of Video for Transmission under Resource Constraints. In *Proc. 17th IEEE Intl. Conf. Image Processing (ICIP98)*, Chicago, Illinois (1998)
7. Sabata, B., Chatterjee, S., Davis, M., Sydir, J., and Lawrence, T.: Taxonomy for QoS Specifications. In *Proc. IEEE Computer Society 3rd International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS '97)*, Newport Beach, California (1997)
8. McCanne, S., Jacobson, V.: vic: A Flexible Framework for Packet Video. *Proc. ACM Multimedia*, San Francisco, California (1995)
9. Chatterjee, S., Bradley, K., Madriz, J., Colquist, J., and Strosnider, J.: SEW: A Toolset for Design and Analysis of Distributed Real-Time Systems.” In *Proc. 3rd IEEE Real-Time Technology and Applications Symposium (RTAS97)*, Montreal, Quebec (1997)
10. Jensen, E., Locke, C., and Tokuda, H.: A Time-driven Scheduling Model for Real-Time Operating Systems. In *Proc. IEEE Real-Time Systems Symposium* (1985)