A QOS SUPPORT FRAMEWORK FOR DYNAMICALLY RECONFIGURABLE MULTIMEDIA APPLICATIONS

Scott Mitchell, Hani Naguib,
George Coulouris and Tim Kindberg

Distributed Systems Laboratory
Department of Computer Science
Queen Mary and Westfield College
University of London

{scott,hanin,george,timk}@dcs.qmw.ac.uk

Abstract: The use of multimedia in distributed systems has begun to include such complex and mission-critical domains as digital television production, 'video-on-demand' services, medical and security systems. These applications impose more stringent requirements on the support mechanisms provided by underlying networks and operating systems than most currently deployed continuous media applications. This paper describes the DJINN multimedia programming framework, which is designed to support the construction and dynamic reconfiguration of distributed multimedia applications. We motivate the benefits of a runtime model of the quality of service and other characteristics of multimedia applications, and demonstrate a generic algorithm for scheduling dynamic reconfigurations that maintains QoS guarantees. QoS characteristics are modelled as piecewise-linear or quadratic relations, which are solved using standard constraint programming techniques. During reconfigurations, updates to active components are scheduled so as to maintain temporal constraints on the media streams. We illustrate our approach using experimental results from a real-world application domain.

Keywords: Components, multimedia, quality of service, dynamic reconfiguration

L. Kutvonen et al. (eds.), *Distributed Applications and Interoperable Systems II*

1 INTRODUCTION

The use of multimedia—or more particularly continuous, real-time media streams—in distributed systems has begun to include such complex and mission-critical domains as digital television production, 'video-on-demand' services, medical applications and security systems. Because of the enrichment they bring to application content we believe that this trend will continue and that more and more distributed mission-critical applications will begin to incorporate continuous media data. These applications impose more stringent requirements on the support mechanisms provided by underlying networks and operating systems than currently more widely deployed continuous media applications such as videoconferencing, streaming audio and video on the Internet and (non-distributed) entertainment software. The quality of the media being presented is important—sometimes critically so—and thus resources must be properly allocated and scheduled in order to preserve this quality. The following three scenarios illustrate some of the problems that will need to be addressed by an application framework for the construction of mission-critical multimedia applications:

- **Digital TV studio.** The production of a digital TV newscast is likely to include: incoming live news footage in a variety of formats; the use of archive material from several sites and in different formats; a news reader (anchor) interviewing remote subjects; frequent changes of programme source on-the-fly. The construction of a system to support such a demanding set of real-time activities while maintaining a continuously high quality of service seems well beyond the capacity of today's digital multimedia platforms.

- **Distributed surgery.** A distributed conferencing system could support a medical team undertaking a transplant operation. The scarcity of specialists makes it necessary to support remote participation in surgical and other procedures. A transplant operation might involve two patients (donor and recipient) undergoing concurrent operations in separate rooms with other specialist consultants participating remotely. Additional channels would provide remote monitoring of patients, remote manipulation of surgical probes, etc. These would also require strong QoS guarantees and consistency constraints. The reliability and quality of service in such an application may be life-critical.

- **Remote surveillance.** A video surveillance system for a major public event (e.g. a political party congress) incorporates a control room accessing the majority of available video and audio sources, but with other agencies supplying and receiving additional streams of information in a variety of formats via land lines and radio. Some of the sources and destinations of audio and video streams are mobile with variable bandwidth and connectivity. Some of the key requirements are to keep certain audio and video channels open to mobile users, to switch transmission links in response to communication failures, and to upgrade the quality of service in order to provide closer observation in response to suspicious incidents.

Applications such as these are often long-lived and subject to frequent reconfiguration and long-term evolution of application structure. The application

software that supports them must be highly adaptable and be capable of tolerating a wide variety of reconfigurations and extensions while still meeting their Quality of Service (QoS) guarantees.

This paper describes the DJINN multimedia programming framework [13], which is designed to support the construction and dynamic reconfiguration of distributed multimedia applications. The main requirements addressed by DJINN are to provide QoS and integrity guarantees for complex multimedia applications, both in their steady state and during reconfigurations. In particular, DJINN includes:

- Programming support for distributed multimedia applications. This includes the means to encapsulate potentially complex configurations of multimedia-processing components, and to abstract away from the details of hardware.
- Dynamic reconfiguration. The requirement is to support dynamic changes to complex component structures, such as when users join and leave groupware sessions. These changes to the application's structure need to be performed atomically, and the application's structural integrity must be maintained—for example, ensuring that the media formats handled by interconnected components are compatible with one another.
- Support for QoS negotiation, admission control and the specification of integrity constraints. This support is available to concurrent applications that can alter their QoS characteristics (e.g. audio quality) at run-time. The QoS support in Djinn provides an environment for adaptable multimedia applications to rapidly converge into a sustainable level of quality.

The rest of this paper is structured as follows. Section Figure 2 is an overview of the DJINN architecture. Section 3 presents an illustrative example of a real application built in Djinn and demonstrates our approach to QoS management and dynamic reconfiguration. Section 0 briefly reviews some related research while Section 5 contains a summary and conclusions.

2 FRAMEWORK ARCHITECTURE

DJINN applications are constructed from networks of *components* consuming, producing and transforming media data streams and interconnected via their *ports*, in a similar fashion to other distributed multimedia programming frameworks such as [2], [8] & [9]. Our approach to meeting the requirements outlined above is based around the use of a *dynamic runtime model* of the application, which models the QoS, structural configuration and integrity properties of the application. The model is itself built from interconnected components, so that DJINN applications have a two-level structure as shown in Figure 1. The active components of an application are autonomous objects that produce, consume and transform multimedia data streams. Active components are distributed so as to meet the processing requirements of the application—in general, they must be co-located with the multimedia hardware that they control. On the other hand, model components do not directly process media data and can be located wherever is convenient for the application user or programmer. The model may be distributed, for example in a video-server system where the server and clients are under the control of different people or organisations.

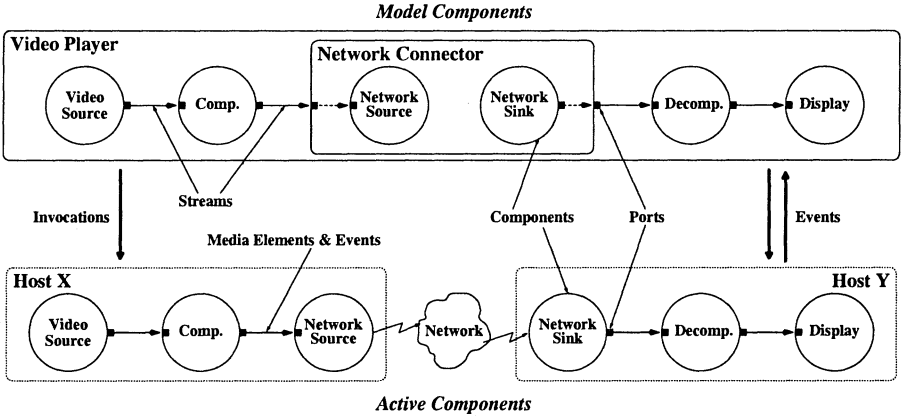

Figure 1. Model and active components.

The model components of an application are arranged in a tree-structured hierarchy, where the leaves of the tree are *atomic* model components, each corresponding to a single active component (for example, the Video Source and Display components in Figure 1). Atomic model components export a common interface to their underlying active components, such that all "Camera" components will offer a common set of operations irrespective of the physical type of camera controlled by the active component. Additionally, atomic model components model the QoS characteristics of their underlying active components as sets of linear and quadratic relations between attributes—such as frame rate and size—of the media streams being processed. These relations include the resource requirements of the active component and any constraints it imposes on the media streams. The connectivity of the active layer is mirrored by the atomic model components: each has the same set of ports and inter-component connections as its active counterpart.

The interior nodes of the model component tree are *composite* components. These components do not correspond to any one active component; rather, they encapsulate a sub-tree of the application model, with the composite component at the root. Composite components facilitate high-level application structuring and add additional behaviour to an application by providing operations to manipulate their encapsulated sub-components. For instance, a video-conferencing component would provide operations to add and remove conference participants. A composite component models the connectivity of its encapsulated sub-tree as a directed graph that can be expanded down to the atomic component level. The root composite component (the Video Player in Figure 1) also stores a *cost-benefit function*, which expresses the application's specific resource/QoS trade-offs.

Application integrity is modelled by sets of predicates attached to model components. Predicates range from simple checks on atomic components—such as ensuring that output ports are only ever connected to input ports—to complex consistency tests on high-level composite components—a video-conferencing component should maintain full connectivity between all participants as well as enforcing a floor-control policy. The predicates are evaluated in leaf-to-root order,

and all must be true for the application's configuration to be considered valid. The bottom-up ordering allows a composite component further up the tree to declare the configuration invalid when it fails to meet a condition unknown to the sub-components.

Application programmers are unaware of the distinction between model and active components. All application-level programming in Djinn takes place at the model layer. Active components are created, configured and destroyed as required under the control of the application model. Components are controlled through a combination of remote invocations and inter-component events. Events can be transferred between components and additionally may flow along the same paths as media streams, interleaved with media data elements. Events enable heterogeneous components to respond to state changes; they also allow us to synchronise reconfigurations with media data flow.

Our primary motivation for the use of an application model is to clearly separate the design of an application from its realisation at run-time [13]. The model is largely independent of location, hardware platform, operating system and the various technologies used to process and transport media data; it enables programmers to build and evolve applications at a high level of abstraction. Active components, on the other hand, have no notion of their place in the larger application—they simply carry out their tasks of producing, processing, transporting and consuming multimedia data.

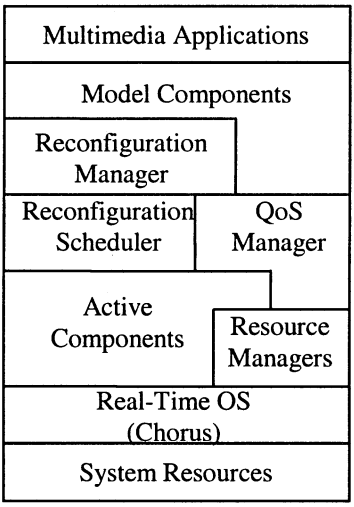

Figure 2. DJINN runtime architecture.

Figure 2 shows the relationships between the main components of the Djinn runtime architecture. The QoS and resource managers provide QoS management support, including admission control and resource allocation. The reconfiguration manager is responsible for controlling and validating changes to the application model; the reconfiguration scheduler maps approved changes onto the active component layer.

DJINN's QoS guarantees depend upon appropriate real-time support from host operating systems and networks. We have a real-time testbed system comprising a set of hosts running the Chorus/ClassiX RTOS [10] and a dedicated Ethernet. Active components on the Chorus hosts are implemented in C++ while the model components—which do not require a real-time platform—are implemented in Java. CORBA is used for inter-component control communication; media streams use protocols appropriate to the stream type and the underlying network.

3 AN ILLUSTRATIVE EXAMPLE

In this section we analyse an application scenario similar to that described by Yeadon et. al. in [22], who are developing systems to provide mobile multimedia support and applications for the emergency services. The setting is a large security-conscious site—such as a factory or research centre—equipped with fixed surveillance cameras feeding video to one or more central servers. Security personnel can monitor the live video streams via either fixed workstations or mobile terminals communicating over a WaveLAN wireless broadcast network [20]. Mobile users who move outside the coverage area of the WaveLAN are still able to receive video over a GSM cellular link [17], albeit with significantly reduced quality. In the event of a major incident—say a factory fire—where the emergency services are called, the surveillance video streams can be routed to the police/fire brigade control room over a high-speed wired link. Relevant streams will then be forwarded to emergency units *en route* to the scene, again using a GSM connection or dedicated packet-radio network. Once on the scene, emergency services personnel should be able to receive the higher-quality video available from the WaveLAN at the incident site. If audio streams are also available, they can be treated in the same way. A high-level view of this scenario is shown in Figure 3.

Clearly this system is subject to frequent reconfiguration as video streams from different sources are switched between the different networks. One of the key requirements of the application domain is for high levels of availability and

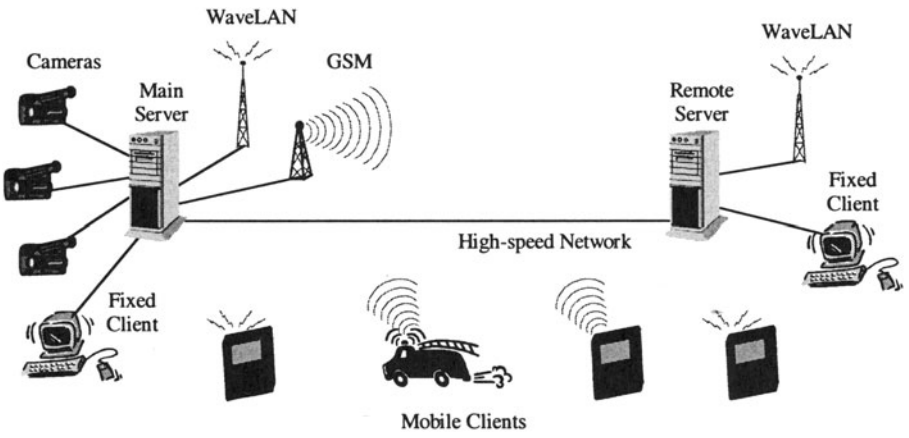

Figure 3. The example application.

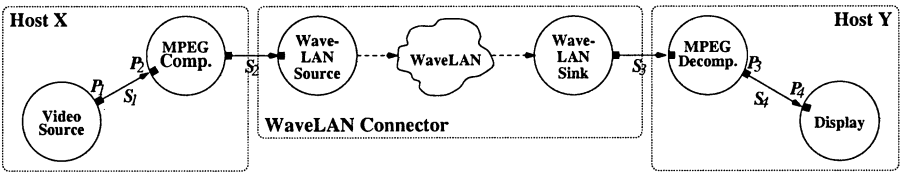

Figure 4. Initial configuration.

dependability of data [22]. This implies a need for seamless switching between network transports at the client end, and careful control of resource usage, especially in highly constrained environments such as the GSM network.

For the purposes of illustration we will consider just one aspect of this application with particular relevance to DJINN: a single mobile video unit that joins the system, then moves from the local WaveLAN to a dialup GSM link. This allows us to address two important aspects of DJINN: First, the admission control mechanisms that allow a new client to join the application with an appropriate guaranteed QoS level; and, second, the algorithms used to schedule a smooth hand-over between the two networks with minimum disruption to the output seen by the user. The initial state of this system is shown in Figure 4.

3.1 Application Setup

Programmers build DJINN applications by creating and interconnecting model components. Before the active components are created and started the model must pass through integrity tests—as described in Section 2—and an admission test. These tests aim to find an application configuration which does not break any of its constraints and for which enough system resources can be reserved. As an example of the former, the main video server in the surveillance application can support a fixed maximum number of GSM connections, determined by the number of attached modems. Any configuration of the model that exceeds this limit must be rejected.

Admission Test. Each admission test utilises the application's QoS model, and is performed in three stages: to gather application-imposed constraints, to determine constraints on resources, and to generate a solution using a cost-benefit analysis. In the first stage components are asked to provide a list of their QoS characteristics (Table 1), expressed as simple numerical relations. This includes the amount of resources required by each component along with any constraints imposed by these components on the streams they process. Consider the remote surveillance example shown in Figure 4. The *Video Source* component imposes the constraint $S_1.rate \leq 30$ due to its frame-rate limitations. The constraint $S_5 \geq 5$ imposed by *Display* is user-specified and ensures that the displayed video will have a frame rate of at least 5 frames per second. The *MPEG Encoder* also imposes constraints on the frame sizes it can produce. Note that to simplify this discussion Table 1 shows only the CPU requirements of components; other resources are treated in a similar fashion.

The QoS characteristics of components are stored within individual model components. The component programmer specifies inter-stream constraints when she creates the component. Our approach to modelling the resource requirements has been to perform direct measurements of these values. We are currently developing a test-harness, which provides the modeller with information related to the component's resource utilisation characteristics. The user wishing to model the component inputs multimedia elements of known attributes (for example, video of known frame rate and size). The harness measures the resource usage. Currently, we measure CPU, memory and network utilisation. We provide a tool for the user to match the resultant data points to linear functions or piecewise linear functions. Sometimes they are functions of products of attributes (for example, frame size times frame rate)—and so we obtain a quadratic function of attributes. Another complication is that resource utilisation may depend on media values. For example, an MPEG decoder may take differing amounts of time to decode two frames of the same type (I, P or B) and size. We therefore can derive several linear or quadratic relations, corresponding, in the case of MPEG, to video of differing classifications [18] (e.g. streams with low level of motion, computer generated animations etc).

Component	Constraints	Resource	Requirement (ms/sec)
Video Source	$S_1.rate \leq 30$	CPU at X	$6.46x10^{-4}S_1.rate*S_1.size$
MPEG Encoder	($S_1.x = 128$, $S_1.y = 96$) or ($S_1.x = 176$, $S_1.y = 144$) or ($S_1.x = 352$, $S_1.y = 176$) or ($S_1.x = 704$, $S_1.y = 575$) or ($S1.x = 1408$, $S_1.y = 1152$)		$1.61x10^{-4}S_1.rate*S_1.size$
WaveLAN Connector	$S_2 = S_3$ (all attributes)	CPU at X	$8.07x10^{-5}S_1.rate*S_1.size$
		CPU at Y	$8.07x10^{-5}S_1.rate*S_1.size$
MPEG Decoder	$S_3 = S_4$ (all attributes)	CPY at Y	$1.08x10^{-3}S_1.rate*S_1.size$
Display	$S4.rate \geq 5$ $120 \leq S_4.width \leq 704$ $80 \leq S_4.height \leq 575$	CPU at Y	$3.22x10^{-4}S_1.rate*S_1.size$

Table 1. QoS Characteristics.

In the second stage of the admission test, relevant resource managers are asked about the availability of their resources. The components' resource requirement functions are turned into a set of inequalities (one for each resource) which express the bound on the resources that can be used by the application. This allows the current resource availability to be expressed within the model. This is shown in Table 2.

The third stage of the admission test attempts to solve the constraint relations. We currently use techniques borrowed from operations research used in optimisation problems. These techniques utilise a benefit function (in our case the application-specific cost-benefit function) to find optimum values for a set of variables (the stream attributes) given a set of constraints (the stream and resource constraints). For our example we use a cost-benefit $f=w_1*S_4.rate + w_2*S_4.size + w_3*(R_{cpuX} + R_{cpuY})$. This is a weighted function (the weights are w_1 w_2 and w_3) of the frame rate and size (which we want to maximise) and the total resource utilisation (which we want to minimise). We use $w_1 = w_2 = 10^6$ and $w_3 = 1$ to express the relative importance of good QoS over resource costs.

These numerical relations are then solved at run-time with the application's benefit function to determine an optimum QoS state. In this example this has a frame rate of 10fps and a frame size of 352x176. This reflects the limited CPU resource availability at host Y. At present we use a freely available linear solver, which limits or models to one stream attribute. We are currently evaluating other more general-purpose solvers, which do not have this restriction.

Resource	CPU Availability (ms/sec)	Resource Constraint
CPU at X	800	$8.877\text{x}10^{-4}S_4.\text{rate}*S_1.\text{size} \leq 800$
CPU at Y	920	$1.482\text{x}10^{-3}S_4.\text{rate}*S_1.\text{size} \leq 920$

Table 2. Resource constraints.

3.2 Dynamic Reconfiguration

We now consider the problem of *reconfiguring* the system in response to a user request or changes in the operating environment of the program. An example of the latter occurs when the mobile handset moves outside the range of the WaveLAN—if video playback is to continue the application must be reconfigured to deliver the video data over the lower-bandwidth GSM network

Application configuration—and reconfiguration—is expressed in terms of paths: model layer end-to-end management constructs describing the media data flow between a pair of endpoints chosen by the application. A path encapsulates an arbitrary sequence of ports and intervening components that carry its data. It declares the end-to-end QoS properties of that sequence, including latency, jitter and error rate. It is up to each individual application to identify the end-to-end flows that are of interest to it and specify paths accordingly. Flows that are not part of a path do not receive any end-to-end guarantees either for their normal operation or during reconfiguration.

A reconfiguration moves the application from one consistent state to another in an *atomic* manner. That is, if it is not possible to successfully perform *all* of the actions required to execute the reconfiguration, then *none* of the actions will be performed and the application will remain in its initial state. The reconfiguration is initially enacted on the application model; no changes are made to any active components until the new configuration has been approved by the admission control mechanism

and validated against any application-defined integrity constraints. If it turns out that the requested changes cannot be successfully applied, the model components are 'rolled back' to their previous consistent state, leaving the application configuration unchanged.

The continuous media streams processed by the active components have constraints that must be maintained *during the transition between the initial and final configurations*. For example, it would not generally be acceptable for the arrival of a new mobile handset in the system to disrupt the video playback on other handsets. Therefore, we apply an ordering or schedule to the active component updates, to maintain the temporal consistency of streams across reconfiguration boundaries, a requirement we have informally named the 'smoothness' condition [14]: *"The execution of a reconfiguration on a live system must not break any temporal constraint of any active path."*

The schedule ensures that the streams will be free of, or at least not unacceptably affected by, 'glitches'. Glitches are lost data or loss of synchronisation, which appears to users as frozen frames, silences or unsynchronised sound and vision.

In our example, the WaveLAN infrastructure is able to detect a change in signal strength indicating that the user is moving outside the coverage area of the network [7],[15]. When this occurs, an event is delivered to the application model causing it to initiate a hand-over to the GSM network. We assume that the WaveLAN can provide sufficient advance notice of an impending loss of service that we can have the GSM link fully up and running in time for a seamless hand-over. The reduced bandwidth of a GSM link (only 9600 bits/s) necessitates a reduction in frame rate and a switch to a more efficient—but lower quality—H.236 codec [5]. Figure 5 shows the final state of the path undergoing the reconfiguration (cf. the initial configuration in Figure 4).

The temporal constraints on this reconfiguration are:

- That the interval between the arrival at P_4 of the last frame from the initial configuration and the first frame from the final configuration is less than 200ms.
- That the play-out times of these two frames should not differ by more than 400ms, i.e. no more than two frames lost or repeated.

Deriving the Schedule. Table 3 shows the latencies and *startup times* for the components in both configurations, where the latter is the time required to get a newly created active component into a state where it is ready to process media data. This is particularly relevant to this example, since the GSM network components

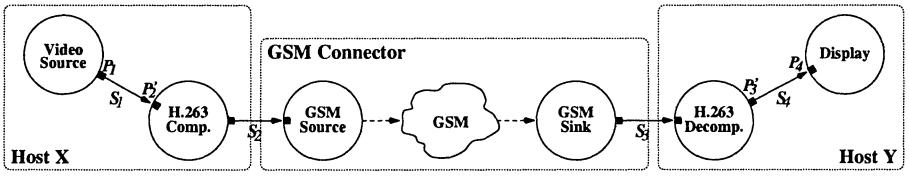

Figure 5. Final configuration.

have startup times three orders of magnitude greater than their operating latency. While the startup delay cannot be avoided, it is possible to reduce or eliminate its impact in the relatively common case that the application receives some advance warning of the need to reconfigure. The achieve this, we divide the active component updates into two phases:

- **Setup.** This phase encompasses the creation of new active components and reservation of their resources. The initial configuration remains operational throughout. However, some of the new components may be started running if the smoothness requirements of the reconfiguration demand it.
- **Integrate.** This phase is started by an event delivered after the end of the setup phase—in our remote surveillance example this event arises when the signal strength reaches a lower threshold. It completes the transition to the final configuration according to a schedule computed to maintain the temporal constraints of the reconfiguration.

Component	Latency (ms)	Startup time (ms)
Video Source	40	500
Display	20	100
MPEG Encoder	100	1000
MPEG Decoder	67	1000
H.263 Encoder	200	1000
H.263 Decoder	100	1000
WaveLAN Source	5	100
WaveLAN Sink	5	100
GSM Source	5	5000
GSM Sink	5	5000

Table 3. Component latencies.

Each active component is 'primed' during the setup phase with the actions to perform during integration. The actions are triggered by receipt of an event from an external source or on an input port; the event is also propagated downstream along the reconfiguration path. Integration is thus performed by scheduled delivery of integrate events to the farthest upstream points of the reconfiguration.

The scheduling algorithm works upstream along both versions of the path from P_4, summing the latencies of each component encountered. When the configurations converge again at port P_1, the differences in latencies along each path allows us to calculate when the last MPEG and first H.263 frames should be delivered to ports P_2 and P_2' respectively. Thus, for the frames to arrive simultaneously at P_4, we should inject the 'start' event into P_2' 133ms before sending the 'stop' event to P_2. We may stretch or compress this schedule by up to 200ms and still meet the first constraint.

Because the *difference* in the latency of the two configurations is less than 400ms, the second constraint is also maintained.

Dynamic Admissions. The above schedule assumes that sufficient resources are reserved, by a dynamic admission test that is part of the atomic action. Dynamic admission tests are slightly different from the initial admission test explained above. The major difference is that these tests must take into account the period during the transition from the initial configuration to the final configuration, where components from both configurations may be executing concurrently. We thus perform two admission tests, one for the final configuration and one for the transitional period.

Dynamic admission tests use the initial state of the model when looking for a solution to the final configuration. The techniques used are similar to those found in sensitivity analysis [12] and can greatly increase the performance of these tests. Furthermore components and resource managers that are not affected by the reconfiguration need not be consulted since their information is already present in the model. This is particularly useful since in many cases it is the QoS characteristics of just a few localised components that are affected. Table 4 shows the time taken to perform admission control calculation with and without re-use of previous calculations.

Number of relations	Complete recalculation (sec)	Re-using calculations (sec)
220	0.20	0.02
1860	2.00	0.18
5100	11.00	0.70

Table 4. Speedup from calculation reuse.

4 RELATED WORK

The component-based approach to application construction is used by a variety of multimedia programming frameworks, such as that of Gibbs & Tsichritzis [9], Medusa [21] and CINEMA [2]. CINEMA also makes use of composite components and a separate 'model' of the application that is used for control and reconfiguration. However, CINEMA's idea of what constitutes a reconfiguration is quite limited and has no equivalent of the 'smoothness' property for ensuring clean transitions between consistent states. It does allow inter-stream dependencies to be taken into account when performing admission control, but it requires application components to be created from the outset in order to provide information about constraints, rather than using a separate model. Also, the application components individually attempt to reserve resources during the admission test. This can lead to admission failing, even in situations where sufficient resources might be found.

The need for smoothness support in the real-world domain of digital television—where there is a requirement to "splice" together MPEG streams within the resource constraints of hardware decoders whilst still meeting QoS guarantees—is illustrated

by [4]. In [19], Sztipanovitz, Karsai and Bapty present a similar two-level approach to component-based application composition in the context of a signal-processing system whose applications share many of the real-time requirements of multimedia.

The use of a QoS model can also be found in the Quorum project [6]. They model the structural and QoS characteristics of applications and use benefit function to capture user preferences, although they do not consider smoothness properties.

5 SUMMARY AND CONCLUSIONS

This paper has motivated the benefits of a runtime model of the quality of service and structural integrity characteristics of multimedia applications. It has also demonstrated an algorithm for scheduling dynamic reconfigurations which maintains QoS guarantees. QoS characteristics are modelled as piecewise-linear or quadratic relations, which are solved using standard constraint programming techniques. The result is a negotiation between the application and the system, with user-configurable bounds. During reconfigurations, updates to active components are scheduled so as to maintain temporal constraints on the media streams. A generic software solver computes the schedule. We have illustrated our approach using preliminary experimental results from a real-world application domain.

A number of issues remain unresolved regarding the utility of our approach. It is not yet clear that resource requirements can always be modelled accurately as piecewise linear or quadratic functions, or that the model is sufficiently generic to be transparently reused in different application domains. In the example presented in this paper we have made some simplifications (in addition to considering only CPU resources). In particular the cost-benefit function should express trade-offs between various streams and between the quality of the application versus its resource requirements. Furthermore, compressed streams would have attributes related to the compression parameters, allowing for further trade-offs between stream quality and resource usage to be expressed.

Likewise, our reconfiguration scheduling algorithm is only fully developed for the single path case—we are still exploring the issues that arise when reconfiguring multiple paths with inter-path dependencies. With reference to the requirements outlined in Section 1, this paper has addressed the reconfiguration and QoS aspects. Further details of DJINN can be found in [13] and our approaches to reconfiguration scheduling and application integrity management appear in [14],[16].

References

[1] ATKINSON M., DAYNÈS L., JORDAN M., PRINTEZIS T., SPENCE S., An Orthogonally Persistent Java, *ACM SIGMOD Record* 25(4), December 1996.

[2] BARTH I., Configuring Distributed Multimedia Applications Using CINEMA, *Proc. IEEE Workshop on Multimedia Software Development (MMSD'96)*, Germany, March 1996.

[3] BELLISSARD L. & RIVEILL M., Olan: A Language and Runtime Support for Distributed Application Configuration, *Journées du GDR du Programmation*, Grenoble, France, November 1995.

[4] BHATT B., BIRKS D., HERMRECK D., Digital Television: Making it Work, *IEEE Spectrum* 34(10), pp 19–28, October 1997.

[5] BJONTEGAARD G., Very Low Bitrate Videocoding using H.263 and Foreseen Extensions *Proc. European Conference on Multimedia Applications, Services and Teachniques (ECMAST '96)*, pp 825–838, Louvain-la-Neuve, Belgium, May 1996.

[6] CHATTERJEE S., SYDIR J., SABATA B., LAWRENCE T., Modeling Applications for Adaptive QoS-base Resource Management, *Proc. 2nd IEEE High-Assurance System Engineering Workshop (HASE97)*, August 1997.

[7] DAVIES N. & FRIDAY A., Applications of Video in Mobile Environments, *IEEE Communications*, June 1998.

[8] FOSSÅ H. & SLOMAN M., Implementing Interactive Configuration Management for Distributed Systems, *Proc. 4th International Conference on Configurable Distributed Systems (CDS'96)*, pp 44–51, Maryland, USA, May 1996.

[9] GIBBS S. & TSICHRITZIS D., *Multimedia Programming: Objects, Frameworks and Environments,* Addison-Wesley, Wokingham, England, 1995.

[10] GUILLEMONT M., *CHORUS/ClassiX r3 Technical Overview* Chorus Systems Technical Report, May 1997.

[11] HÄRDER T. & REUTER A., Principles of Transaction-Oriented Database Recovery, *ACM Computing Surveys* 15(4), 1983.

[12] HILLIER F. & LIEBERMAN G., *Introduction to Operations Research.* McGraw-Hill International Editions, New York, USA, 1995.

[13] MITCHELL S., NAGUIB H., COULOURIS G. & KINDBERG T., A Framework for Configurable Distributed Multimedia Applications, *3rd Cabernet Plenary Workshop*, Rennes, France, April 1997.

[14] MITCHELL S., NAGUIB H., COULOURIS G. & KINDBERG T., Dynamically Configuring Multimedia Components: A Model-based Approach, *Proc. 8th SIGOPS European Workshop*, Sintra, Portugal, pp 40–47, September 1998.

[15] MOURA J., JASINSCHI R., SHIOJIRI H. & LIN J., Video Over Wireless, *IEEE Personal Communications* 3(1), pp 44–54, February 1996.

[16] MITCHELL S., NAGUIB H., COULOURIS G. & KINDBERG T. Modelling QoS Characteristics of Multimedia Applications, *Proc. 13th IEEE Real-Time Systems Symposium (RTSS '98)*, Madrid, Spain, December 1998.

[17] RAHNEMA M., Overview of the GSM System and Protocol Architecture, *IEEE Communications Magazine* 31(4), pp 92–100, April 1993.

[18] SHEN K., ROWE L. & DELP E., A Parallel Implementation of an MPEG-1 Encoder: Faster than Real-Time, *Proc. SPIE Digital Video Compression: Algorithms and Techniques*, San Jose, CA, USA, February 1995.

[19] SZTIPANOVITS J., KARSAI G. & BAPTY T., Self-Adaptive Software for Signal Processing: Evolving Systems in Changing Environments without Growing Pains, *Communications of the ACM* 41(5), pp 66–73, May 1998.

[20] TUCH B., Development of WaveLAN, an ISM Band Wireless LAN, *AT&T Technical Journal* 72(4), pp 27–37, July/August 1993.

[21] WRAY S., GLAUERT T. & HOPPER A., *The Medusa Applications Environment*, Technical Report 94.3, Ollivetti Research Limited, Cambridge, England, 1994.

[22] YEADON N., DAVIES N., FRIDAY A. & BLAIR G., Supporting Video in Heterogeneous Mobile Environments, *Proc. Symposium on Applied Computing*, Atlanta, GA, USA, February 1998.