Towards automatically configurable multimedia applications

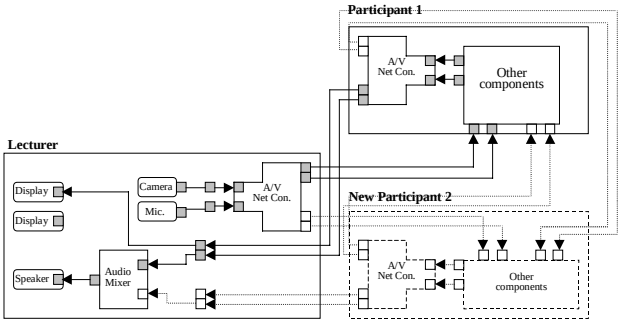
Hani Naguib, George Coulouris
Laboratory for Communications Engineering
Cambridge University
Trumpington Street, Cambridge CB2 1PZ, UK
{han21,gfc22}@cam.ac.uk

ABSTRACT

We describe and illustrate an approach to the automatic configuration of component-based multimedia applications. The approach is based on the deployment of a run-time application model that mirrors the active application components, enabling changes to the configuration to be applied and evaluated in the model before they are deployed. The model employs a composite component structure, enabling complexity to be concealed except when detail is required. Constraints and QoS specifications are embedded in the model.

1. INTRODUCTION

Most multimedia programming frameworks have adopted a component-based approach to application construction. In this approach applications are viewed as sets of interconnected multimedia components. These components being software abstractions of multimedia devices such as cameras, video displays and hardware/software codecs. They generally have a multimedia interface through which multimedia data enters/leaves them, and may also have a separate control interface that allows them to be configured and controlled. Diagram 1 depicts an electronic lecture constructed as a collection of such components.



[Figure 1- An Electronic Lecture]

This component based approach has a number of attractive features:– the components can be made to directly reflect the structure and connectivity of existing (analogue) multimedia applications and applications can be constructed using largely 'off-the-shelf' components. But the approach does not, by itself, guarantee easy application development and maintenance.

In this paper we focus on the configurability of component-based multimedia applications; how networks of components can be setup to meet application requirements and modified in response to changes in the application's context. Section 2 details the problem we are looking at and defines a number of requirements. Section 3 outlines our proposed approach to meeting some of these requirements and illustrates this with an example. We conclude with a brief description of our achievements and further research we intend to undertake in the near future.

2. Configuration requirements

Even relatively small applications such as the one illustrated in Figure 1 are often composed of many components with streams of multimedia data flowing between them. The QoS and type characteristics of the streams can have complex and interrelated interactions. All of these characteristics have to be specified in the course of constructing or reconfiguring an application. In most cases this task falls on a person who takes the role of *application manager*, and who must devise ad-hoc techniques, which are often application-specific and laborious.

In fact there are situations where it becomes almost impossible for the application manager to deal with these problems unless she can access and comprehend low-level system attributes and the intricacies of the components in the application. For example, in our electronic lecture application (Figure 1); the audio/video encoding used by the *A/V Network Connector* may depend on the network and CPUs loads at the various participants. Choosing an encoding scheme requires the manager to reason about system resources and be aware of the resource requirement characteristics of the various encodings that are available, as well as any user defined requirements.

Composite components (components that hierarchically encapsulate complexes of sub-components) can go some way towards reducing this complexity, since they can act as black boxes hiding the complexity of closely related components, such as the A/V Network Connector which, might include various components capable of numerous encodings and transmission schemes. But there is also a need to provide some automatic support for the maintenance of the relevant constraints when application configurations have to be changed. Below we list the most pressing requirements:

Hiding as much of the complexity and intricacies of (re)configurations as possible from both application programmers and users.

A systematic and standard way to specify the relevant characteristics of the application that allows for the system to reason about them and perform the relevant actions needed to enforce them.

3. A model-based approach

We have suggested in the preceding paragraphs that high-level requirements should be explicitly represented and respected when applications are modified. How is this to be achieved? High-level representations and management of constraints do not sit well with the requirements of multimedia processing components. These are generally carefully-tailored for optimum execution speeds, often as a part of the embedded system found in cameras, displays, etc. There appears to be a need to separate the representation of those parts of the application that can be used for reasoning about quality of service and other constraints from the components that process media data.

In this paper we describe the approach taken in QoSDREAM, an object-oriented framework for the development context-aware multimedia applications that we are developing [1]. A key feature of our approach to tackling the problems outlined above is to construct multimedia applications in terms of two separate structured collections of objects which we call the "model layer" and the "active layer". The model layer contains the information that describes both the structure of the application and its QoS characteristics (which include the requirements previously mentioned). The active layer on the other hand contains the code used when the application is active. In other words it is the part of the application that actually produces, processes and consumes the multimedia data. This separation is not directly visible to application programmers or managers, and therefore does not add any unnecessary complexity to the application development cycle.

In QoSDREAM programmers build or modify multimedia applications by creating and interconnecting model components. Before the active components are created and started the model must pass through an integrity test (which checks that they have been correctly interconnected) and an admission test (which configures the various stream attributes, taking into account application characteristics and system resources). If these tests are successful then an application configuration is found which does not break any of the application's constraints and for which enough system resources can be allocated. A schedule for updating the application's active layer to reflect the new configuration is calculated and the changes take effect.

Integrity checks. Application integrity is modeled 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 – the electronic lecture application should maintain full connectivity between participants and the lecturer as well as enforcing a floor-control policy. The predicates are evaluated in a 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.

Admission Test. Each admission test utilizes the application's QoS model. This model reflects the QoS characteristics of the application and includes the following information:

Resource requirements of the components in the application.

Functional constraints imposed by the components in the application.

User defined constraints that reflecting the requirements of the user.

A user-configurable benefit function which describes the characteristics of the application that the application user finds

The functional constraints are expressed as numerical relations between the various attributes of the streams that a component processes. For example a camera may have a functional constraint that limits the range of frame rates it is able to provide. This might be expressed as 0 <= VideoOut.rate <= 30. The resource requirements are also expressed by similar numerical relations, for example the camera's resource requirement is expressed as a function of the frame-rate and size of the video it is to capture. The value for the resource requirements is currently obtained by direct measurements and can be refined dynamically. User defined constraints are similar to functional constraints except that they are imposed by the application user and are not inherent in the components in the application. An example of such a constraint may be to impose a lower bound on the video's frame rate. The benefit function is a weighted function which expresses the relative importance of the various stream attributes over resource costs.

The reason for expressing the QoS characteristics of applications as numerical relations is that it provides a standard approach to this specification and allows the system to automatically reason about and perform admission tests. We currently use techniques borrowed from operations research used in optimization problems, coupled with some of our own techniques, such as restricting the acceptable values for stream attributes based on existing standards to simplify this model in order to obtain feasible solutions and thus obtain values for the stream attributes, giving us an acceptable application configurations.

Scheduling Active Layer Updates. The actions needed to complete a reconfiguration can be quite time-consuming, especially if new components must be deployed and activated (this may involve setting up hardware devices as well as considerable amount of remote invocation). However, in many reconfigurations there is a requirement that the transition between the old and new configuration takes place either at an absolute real time or within some interval of someone "pushing the button" – both are difficult to achieve if there is an indeterminate amount of setup activity to perform before the switch can take place. We generate and conform to a configuration schedule to overcome this problem. A description of how this schedule is calculated is beyond the scope of this paper, please refer to [3] for a detailed description.

To ensure the atomicity of (re)configurations, which ensures that applications are always kept in a consistent state, all of the above tests are performed within a form of transaction that we call multimedia transactions [2]. If for any reason the test are not successful in finding an adequate configuration the model can be rolled-back to its initial state.

4. Example

To demonstrate how the reconfiguration process is achieved we will look at the introduction of a new participant in our electronic lecture example. Due to limitations on the size of this paper only an overview is given. For more detailed information please refer to [3].

```
Participant Lecture::AddParticipant(String host) {
   trans.begin();
   newParticipant = new Participant(host);
   for (Enumeration e=existingParticipants.elements();
                                          e.hasMoreElements();)   {
      nextPar = (Participant) e.nextElement();
      newParticipant.connectTo(nextParticipant);
   }
   newParticipant.connectTo(lecturer);
   existingParticipants.addElement(newParticipant);
   trans.commit();
   return newParticipant;
}
```

For the application developer, adding a new participant into the lecture requires only the invocation of a single method (*Lecture.AddParticipant(host)*). This method initiates a multimedia transaction, creates a new participant object and connects the new participant to all other existing participants and the lecturer.

The creation of required components is automatically made by methods within participant. For example invoking the *connectTo* method in the *Participant* object causes the creation of the audio/video components needed to support a new participant.

Most of the required information needed to perform the integrity and QoS tests needed during (re)configurations is already present at this stage. The interconnectivity of components is visible by the interconnections performed by the composite components (remember that composite components know how to interconnect their top-level sub-components).

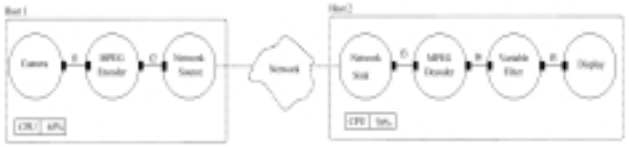
The QoS characteristics of individual components are also known (since this information is part of the QoS Model). The only missing parts are user-specific QoS requirements, system resource availability and an overall cost/benefit function for the application.

System resources are made available through *Resource Managers*. Resource Managers are components responsible for monitoring and allocating system resources. There is one resource manager per system resource. Shared resources such as network bandwidth is generally controlled by sets of co-operating resource managers. It is important to note that if guarantees on system resource availability are to be given to applications then the underlying operating system needs to provide adequate real-time support. For example the ability to reserve CPU cycles to specific threads or applications and to schedule its usage in a manner appropriate to multimedia applications (i.e. deadline based scheduling of multimedia threads).

The user's QoS requirements are specified through methods exported by components. For example *Display.setFrameRateRange(lowest,optimum)*. Components are in charge of translating these requirements into the appropriate mathematical relations used during the QoS tests.

The overall configuration quality measure is also made available to users through exported component methods. These are usually made visible through GUI controls such as sliders, giving the application user the ability to express the weighted importance to be given to various application quality measures. For example expressing the relative importance of the audio versus the video parts of a transmission. In the lecture example more weight will probably be given to the audio stream. This might be reversed in the case of a transmission of a sporting event. Components that export such interfaces then convert the user's preferences into weights used in the benefit function. Application programmers can also make this option available through pre-set quality measure templates, in much the same way existing audio streaming applications allow users to specify their bandwidth availability, which dictates the quality of the stream they receive.

To illustrate how the mathematical model is used to find appropriate application configurations we will look at a very simple application, where a single video stream is being transmitted. The model for the lecture example is to complex to be properly discussed in this short paper.



[Figure 2 - Example Application]

In this example (Figure 2), we vary only the frame rate, and assume that we want to maximize the frame-rate arriving at Display (benefit function = max f5). We model the MPEG encoder and decoder's CPU resource requirements using values obtained by direct measurements, in such a way that we aim to drop no more than 10% of any type of frame. In a complete model all resource are taken into account and differing drop-rates are allowed for I, B and P frames.

Table 1 shows the resource requirements of the components found in this example.

Component	Constraint	CPU Requirements (micro secs/sec)
Camera	f1<=30	200*f1
MpegEncoder	f1=f2	25000*f1
NetSource	f2=f3	1700*f2
NetSink	f3=f2	100*f3
MpegDecoder	f3=f4	4000*f3
Var.Filter	f4>=f5	10*f4
Display	f5>=10	2000*f5

Table 1- Model of component's QoS characteristics.

During admission test the resource availability is also found by asking the appropriate resource managers. Table 2 shows these values (for this example) as well as the inequalities that must be added to the model in order to ensure that the application can obtain its' required resources.

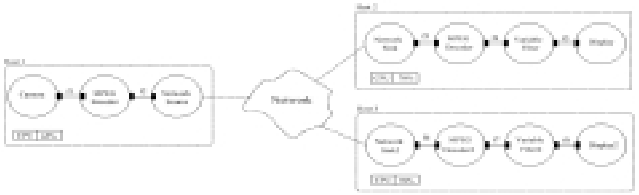
Variable Filter components play an important part within the procedure of finding acceptable configurations for the application. They can modify the value of stream attributes (in this case the frame-rate, by dropping frames). This de-couples the values of downstream stream attributes from those upstream, allowing for greater flexibility. For example if this video were being sent to more than one destination, Variable Filter would allow for the two end points to have different properties. In the case of the lecture

example, Participants could each specify differing qualities for the streams they receive.

To illustrate this we modify the example by introducing a new sink for the video stream, this is depicted in figure 3.

Resource	CPU Availability (micro secs/sec)	New Inequalities
CPU1	650,000	200f1+25000f1+1700f2<=650000
CPU2	700,000	100f3+4000f3+10f4+2000f5 <=700000

Table 2- Inequalities constraining resource usage



[Figure 3 - Modified Example]

The QoS characteristics of the new components are shown in table 3. Table 4 shows the CPU availability and new inequalities found at host 3.

Component	Constraints	CPU requirement (micro secs/sec)
NetSink2	f6=f2	100f6
MpegDec2.	f6=f7	4000*f6
VarFilter2	f7>=f8	10*f7
Display2	f8=15	2000*f8

Table 3- New QoS characteristics

Resource	CPU Availab. (micro secs/sec)	New inequalities
CPU3	780,000	100f6+4000f6+10f7 +2000f8<=780000

Table 4- New Inequalities constraining resource usage.

If we assume that the model for this example was constructed due to a reconfiguration of the initial example, then we can use existing techniques that allow for solving a model from an existing model. Since reconfigurations usually leave much of the application untouched this approach can in many instances decrease the time required to solve models. Table 5 shows the time taken to solve the relations for a number of sample large applications, both with and without the reuse of the previous state of the model.

It is important to notice that most of the work of configuring the application has been done automatically. The application programmer needed to connect a relatively small number of components (they are only aware of the top-level composite components), and specify any user requirements along with a benefit function and any integrity constraints he/she may wish to impose. The integrity checks ensure that components are connected in a consistent manner. The admission test configures the application, finding suitable values for all stream attributes.

#Relations to resolve	Time to solve	Time to solve (reusing model)	Speedup
220	0.2	0.02	10
1860	2	0.18	11.1
5100	11	0.7	15

Table 5- Speedup from reuse of models.

5. Conclusion - Further work

Although this paper is brief we hope we have conveyed the requirements for middleware support of configurable multimedia applications and our approach to tackling them. The model/active layered representation of applications allows great flexibility in the type of tests that can perform without significantly increasing the complexity to application programmers. The modeling of the QoS characteristics of applications as mathematical relations allows the framework to reason about the application and to automatically perform many of the tasks that would otherwise fall to application programmers and users. The usage of composite components which export methods that allow the programmer to express her requirements provides a suitable location for translation of high-level user requirements into the more detailed model required, as well as acting as black-boxes, hiding the application's inner structural details.

One important aspect of (re)configuration of multimedia applications that has not been touched in this paper is that of application context. Our research from which the ideas discussed in this paper is taken has been investigating this, in particular the incorporation of location information into a distributed multimedia environment. Please refer to [1] and our web site (http://www-lce.eng.cam.ac.uk/qosdream) for a more comprehensive description of this research as well as the material presented in this paper.

Acknowledgements

The QoSDREAM project is funded by the UK EPSRC under its Multimedia and Network Applications research programme. Heather Liddell, Tim Kindberg and Scott Mitchell also made substantial contributions to the development of the approach described in this paper:.

References

[1] Scott Mitchell, Hani Naguib, George Coulouris and Tim Kindberg, "A QoS Support Framework for Dynamically Reconfigurable Multimedia Applications". In Lea Kutvonen, Hartmut König and Martti Tienari (eds), Distributed Applications and Interoperable Systems II, pp 17-30. Kluwer Academic Publishers, Boston, 1999. Also in Proc. DAIS 99.

[2] R.S. Mitchell, "Dynamic Configuration of Distributed Multimedia Components". Ph.D. Thesis, University of London, August 2000. http://www-lce.eng.cam.ac.uk/qosdream/publications/

[3] Hani Naguib, Tim Kindberg, Scott Mitchell, and George Coulouris. "Modeling QoS Characteristics of Multimedia Applications." Proc. *RTSS 98*, the 13th IEEE Real-Time Systems Symposium, Madrid, Spain, December 1998