Abstract

This paper describes the configuration of distributed multimedia applications using the CINEMA system. Components, interconnected by links via ports are used to handle multimedia data. Configuration management controls these interconnections, establishes components in threads on different systems and allows an interactive control of data handling functions. In this paper, we present a generic application of the CINEMA system to do an interactive configuration of applications using a graphical interface.

1. Introduction

Multimedia handling and the information highway are considered as the future technologies used to build and design new applications and to communicate with each other. Distributed multimedia applications must deal with periodic data streams and hold hard time constraints. A programmer of such a system must handle this great complexity. In this paper, we separate these problems into application specific and generic problems. Handling data units of a data stream is an application specific problem, while data transport, scheduling, and resource handling can be treated in a generic way. Data handling can be supported by providing component concept, where components can be reused in different applications. A guarantee of quality of service in multimedia applications is not only necessary to satisfy users.

This guarantee is needed to assure the semantics of multimedia data. Therefore, reservation of resources is considered to be a necessary action to prevent the systems from being overloaded and an uncontrolled degradation of quality. Further more, multimedia applications often consist of different parts which are used in a sequential order. Reservation of resources for all of these parts in advance would result in underestimation of resources or lead to an overbooking, that can cause problems when executing too much in parallel.

Besides the advantage of a better resource utilization, configuration of distributed multimedia applications allows an easy reuse of media handling functions and the concentration on new multimedia handling functions without looking at communication problems. CINEMA (Configurable INtegrated Multimedia Architecture) is a middleware system that provides high abstractions to build up distributed multimedia applications. These applications can be configured at runtime and allow a dynamic change in the used topology. CINEMA assists the client in the configuration of a media handling topology, the synchronization of media streams, and the resource reservation in an integrated way.

It provides a proxy interface to the client, which handles all aspects of distribution in a way that is transparent for the client. To demonstrate the flexibility and power of the defined system, a general client to build up and control topologies using a graphical interface has been developed. Constructing distributed applications out of simple components by configuring the interconnection links. These projects deal with conventional applications and do not address the problems involved by periodic multimedia data. With multimedia data additional and new problems arise. They concern periodicity, quality of service and resource demands, and the separation of data handling and control.

The idea of configuring distributed multimedia applications out of components is used in a lot of other multimedia projects, too. A great difference between these projects is the role of the components within the topology. In paper, we defined active objects, called components, that can be connected by ports and links.
These active objects include the timing needed for periodic multimedia data. In CINEMA, components are passive objects that concentrate on data handling functions without realtime dependencies. CINEMA components are activated by the system environment according to available data units, to time dependencies of synchronization and to period of data. An activation set defines the relationship between components activity and available data units.

A component is a software entity that capsules a stream handling function or a hardware device. Passive objects offer more flexibility for synchronization control (e.g. for the adaptive synchronization protocol ASP, as adaptive changes in the rate can be performed without changing the component. In contrast to this, a function to adapt the rate must be provided by active components, and component programmers are responsible for the correct behaviour to allow these changes. In the following parts paper, paper will first give a brief overview on the abstractions the CINEMA system provides to clients. Then in the paper present the configuration management of the CINEMA system and explain concepts and algorithms that are used to handle dynamic configuration. This is followed by an example to build up a distributed multimedia application using a client program. The general client, the CINEMA Application Editor, is presented.

Nomenclature: Database Management

2. Abstractions of a configurable multimedia application

In this section we describe the main abstractions of a configurable multimedia application of CINEMA. A distributed multimedia application is defined as a client and a topology of interconnected components. The client builds up and controls the topology. The topology is performing the handling and communication of multimedia data. A component is the basic abstraction for handling multimedia data. It consumes one data unit at a time provided by an input port, executes a specific operation on this data, and offers the manipulated data to the next component using an output port. The ports are interconnected by links that provide a transport service for data units. Links may be realized locally by interprocess communication mechanisms or remotely by transport connections.

The client builds up his topology by selecting the desired components and interconnecting them by defining the links between output and input ports of components. Besides the stream handling functions, the temporal properties of the topology must be controlled, too. As components are defined to be passive objects, a media clock can be added to a port to control the temporal properties of a media stream. A clock hierarchy can be defined by connecting clocks using edges to group media streams and for controlling groups of streams. Operations on a media clock are propagated in a root-to-leaf direction through the clock hierarchy.

A synchronization relationship between streams is defined by reference point, an attribute of an edge from the clock hierarchy. A media clock is normally attached to a port of a sink and controls the data flow from source to sink. Synchronization modules using different synchronization protocols control the data flow. The topology describes the interconnections between the components, the clock hierarchy expressing synchronization relationships, and a session. Sessions are defined to select parts of the topology, that shall be established at a time and that must hold a given quality of service. Data handling of a media stream can only be started, if all components of the media stream are part of an established session.

3. Architecture of configuration service

The configuration service is divided into two parts. One part, serving the client to build up his topology and to do all checks on topology changes (logical representation), is located on the system the client is established and is called application handler. The other part consisting of configuration handlers, is distributed over the network and provides the facility to build up, control and destroy physical components located in threads (physical representation). In Fig. 1 this architecture is shown using a simple topology example.

The logical representations of the topologies are managed in application handlers. One application handler deals with all topological information of one client. This separation of clients allows an independent execution of client commands in parallel.
The physical representations of topologies are managed by configuration handlers. On each node in the system, such a configuration handler is installed. All configuration commands are executed by these handlers. Configuration commands can be triggered by different application handlers (e.g. one is establishing a session while another is deleting one). The configuration process is divided into two steps: First, the client defines his topology and the configuration is checked syntactically and semantically. The second step is initialized when the client defines a session and the topology is physically established. In this second step all components necessary for the session are selected and established on the assigned systems. The links needed to allow data flow are build up and all mechanisms necessary for synchronization are prepared. After creation of the session, data handling can be initiated by starting the associated clock.

A session is only a logical object, that contains all source and sink ports, that are defined to be active. This object is not stored anywhere. Whether a component is part of a session or not can be determined by looking whether the component is physically represented or not. With this semantics of sessions allow unrestricted building up and destroying of sessions. Especially, no ranking of session creation and session deletion exists. This allows dynamical changes in the topology and in the activated parts of it.

In the following parts we present the algorithms to
• distribute components on threads,
• select all components that are part of session creation,
• select all components that are part of session deletion.
The simplest way to place components into threads is to assign one thread per component. As multimedia applications very often are interactive, a short delay must be guaranteed to get an acceptable presentation quality. The delay is summed up from communication delay, data handling time and scheduling delay. It must be aimed at reducing these delays as much as possible. The communication delay can only be reduced by using a more powerful network. The data handling time is part of the scheduling time and can only be reduced by using a more powerful system. The scheduling delay depends on the scheduling algorithm and, for the algorithms normally used (rate monotonic and earliest deadline first), on the period of the data stream (a linear reduction can be achieved by using modified algorithms like in )

If each component is scheduled on its own, the delay will be \( N \) times the period if \( N \) components are used in a linear path through the topology. To reduce this delay, multiple components can be placed into one scheduling unit (thread) whenever possible. This placement can be done, when the topology has been defined and a session is in process of establishing. Considering this will lead to a great scheduling delay in a greater topology. As a consequence, delays can be reduced by assigning components, which are linearly connected, into one thread. A simple algorithm will be presented to place components into threads.

The algorithm works in the following way: Starting at the sinks, we check whether the components \( c1 \) and \( c2 \) are located on the same system and whether all input ports of \( c1 \) are connected to ports of \( c2 \). If these conditions hold and \( c2 \) has no connections to other components, \( c1 \) and \( c2 \) can be placed into one thread. Otherwise, \( c1 \) and \( c2 \) must be in different threads. The algorithm is applied to all components connected to \( c2 \) until the sources are reached. To allow a better scheduling of sinks, they are always placed into an own thread. It is obvious that all connections concerning a session must be defined before the components can be installed. Besides this placement into threads, an optimized placement of components on different hosts can be considered.

This topic is beyond the scope of this paper. Dynamic configuration means that new components and links can be added while parts of the topology are already running and other parts will stay in non running state. To achieve this, an algorithm is needed to find all components that must be established for a session, so that the client does not need to provide the necessary information. The client must specify all source and sink components (using the ports) as an input for this algorithm. The basic idea of this algorithm is to mark all components of the topology, that are reached by data streams from the given sources and which deliver information to the sinks. This is achieved by starting at the sources as well as sinks and marking all components of the topology following all output resp. input ports.

Only those components carrying both marks are needed for the session. By also marking the links, all needed connections are known, too. For the selected components, the thread distribution is performed (looking at connections not used for this session, too) and the components and links are established. In Fig. 2 an example for a partially established session and the result of the marking algorithm is shown. Every extension of an existing session is possible with this algorithm. To delete a session (completely or partially) we need an algorithm to detect all components that are involved in other sessions. Looking at the topology, we find
Figure 2. Selecting components for a session

that the problem is the same as the problem of which components are needed after the removal of the session. This problem can be solved by applying the algorithm for establishing the session to all sources and sinks, that are part of the activated topology and not part of the session. With this algorithm all components that are reachable from one of these remaining sources or sinks and have the same marks as these sources or sinks are needed after the deletion of the session. Fig. 3 shows an established session, where the marked components must not be deleted. The corresponding marks from the algorithm are also shown. It can be concluded, that the architecture of the configuration management in CINEMA and the algorithms presented in this chapter enable dynamic configurations of distributed multimedia applications, especially growing and shrinking of these applications.

4. User interface to build up distributed multimedia applications

A user interface to a system platform like CINEMA should be easy to use and simple to understand, otherwise there will be only a restricted use for it. We have defined such an interface based on an object-oriented approach. The client defines his topology using object classes to create components, ports and clocks. Components are specialized by providing the component type name to the constructor of the component object.

Using the component object, the client can access ports of the component and methods provided by it. All methods of such component objects are transparently forwarded to the configuration management. Connecting ports of components by links is performed by a special method provided by input ports.
This method can be invoked at most one time and cares for connection to an output port. Because every input port must be connected to exactly one output port, a separate link object is not needed. Clocks are created from the clock class. They are attached to ports of components by means of another port specific method. A clock hierarchy is constructed by using edge objects and binding a parent and a child clock to it. The edge object is defined as a first-class-object as it provides different attributes (like reference points, edge type etc.).

Quality of service description is located at the input ports of sinks. A port specific method allows the setting of single values or value ranges for the defined streamtype parameters. These parameters are negotiated according to the capabilities of components and availability of resources. The following example shows a simple client program in C++-like pseudo-code including all features as described above:

```cpp
1  CompCamera = new COMPONENT("camera", 2 system1);
2  CompDisplay = new COMPONENT("display", 4 system2);
3  DispPort = CompDisplay->Port("video");
4  CamPort = CompCamera->Port("video");
5  DispPort->Connect(CamPort);
6  clock = new Clock();
7  DispPort->Associate(clock);
8  DispPort->SetQoS(Rate, 15, 25, Picturewidth, 200, 400,
9  Pictureheight, 150, 300, Delay, 150, 250);
10 CreateSession(CamPort, DispPort);
11 clock->Start();
12 CompCamera->
13 Method(SetBright,Brightness, 40);
14 DeleteSession(CamPort, DispPort);
```

The interface is provided as a C++-library which must be linked to the client program. This library is independent from the used components. We defined a generic interface to components that is specialized at run time with information provided by the configuration management. With this interface we can add new components and applications that use these components to an already running CJINEMA system.
5. CINEMA application editor

An important advantage of the independence of the client library from used components is, that programming of clients and of components are independent and can happen at different times. To demonstrate the flexibility of the CINEMA system we have developed the CINEMA application editor. This client program for CINEMA allows the definition of topologies using a graphical X11/Motif-based interactive front-end. When starting the CINEMA application editor, a window pops up, where the user can place his components and connect ports with links etc. A new component is created by defining the component type, the desired location and optionally by a unique name for the component (see Fig. 4). After these definitions have been finished, the component is displayed in the window at the position the user has defined. The component is automatically configured with its ports (input ports at the left side, output ports at the right side). To define a link between ports, the ports must be selected with the mouse. Furthermore, the port menu allows to add a clock to the port. With a popup dialogue, the clock can be configured and will be displayed together with the port symbol. A free clock (not attached to a port) can also be created on the desktop.

An edge between two clocks can be defined starting at the child clock and selecting the edge type. The edge is drawn to the next selected clock. The edge-selection is part of the clock menu attached to the clock symbol. Port objects are typed by streamtypes which define the possible quality of service values. These values can be specified by the user in a dialogue. Users can define lower and upper bounds for every variable in the streamtype definition.

Intervals are needed to enable negotiation of quality of service when establishing the session. Components offer methods to control their data handling function. These methods are specific for a component and provide different parameters (in number and type). Methods can be called by users by selecting the desired method from the method menu and setting parameter values.

After execution, the result of the method is displayed in the dialogue window. Clocks and clock hierarchies can be controlled using the clock menu and the dialogue pop-ups when selecting a menu point. Reference points to synchronize streams are defined by two time values, one for each stream. Propagation of commands can be locked using the root of the subhierarchy and selecting clock command that may not be applied to this subhierarchy. Clock commands can be executed using an absolute start time or immediately.

After all objects have been defined, a session is created (or deleted) by selecting input ports of sink components and output ports of source components. These ports define the subtopology used for session creation or deletion. To start session creation/deletion, no additional parameters are needed.

All definitions for quality of service and component initialization must have been made before. If the session could be established/deleted successfully, the component menu of all
activated/deactivated components changes to allow/disable component methods. Fig. 5 shows a screenshot of an example usage of the CINEMA application editor including pictures of executed components. A method Rotate has been called to rotate the contents of the video by an angle of 50 degrees. The displayed windows are generated by the XWindow components.

The video input is handled by the BW8BitVideo-Source component and the picture rotation is done by the Rotate8Bit component. Topologies built up with the CINEMA application editor can be stored and reloaded. A skeleton for a new application using such a topology can also be produced with the editor. Therefore, the editor is more than only a tool to present CINEMA in a graphical way, but it can also be used to do interactive testing of applications before they are programmed as stand-alone applications.

6. Summary and conclusion

We have presented an overview over the configuration management of the CINEMA system. With this system, distributed multimedia applications can be configured dynamically. The configuration is dynamic with respect to used components, connections defined and activated parts of a topology. The distributed architecture of the management system combines the advantages of a centralized handling of the topology at the client's site and the distributed support for the runtime-system.

Algorithms that have to traverse the topology can be executed within the centralized configuration. The results of these algorithms are forwarded to the runtime-system to execute the desired operations. The separation of logical and physical handling of the topologies enable an independent management of clients and at the same time an integrated execution for all topologies in the system.

ACKNOWLEDGMENTS:
Prof. S. H. Patil (H. O. D.), Prof. S. D. Joshi, from Bharati Vidyapeeth Deemed University Pune help for improve the strength of material for this paper.

References


