An adaptive architecture for presenting interactive media onto distributed interfaces

AN ADAPTIVE ARCHITECTURE FOR PRESENTING INTERACTIVE MEDIA ONTO DISTRIBUTED INTERFACES

Jun Hu
Media Interaction Group
Philips Research Laboratories
Prof. Holstlaan 4, 5656AA Eindhoven, The Netherlands
hu@natlab.research.philips.com

Loe Feijs
Dept. of Industrial Design
Eindhoven University of Technology
Den Dolech 2, 5600MB Eindhoven, The Netherlands
l.m.g.feijs@tue.nl

ABSTRACT

This paper introduces an adaptive architecture for presenting interactive timed media onto distributed networked devices. The architecture is put into the test in a storytelling application for children. The interactive story is documented in StoryML, an XML-based language, and presented to multiple interface devices organized in an agent-based architecture. This allows the separation of the content from concrete physical devices, the definition of abstract media objects and the automatic adaptation of the same content to different environments of physical devices. Since both the content and the interaction are timed, issues of streaming and synchronization in this architecture are also addressed.

KEY WORDS

Distributed media, distributed interfaces, architectures

1. INTRODUCTION

For many years, the research and development of timed media technologies have increasingly focused on models for distributed multimedia applications [3,6]. The term "distributed multimedia" refers to the fact that the content sources of a timed media presentation to the final user are distributed over a network. This paper has a different focus which is about distributed presentation interfaces in the user's home. Rather than the distributed content that may be related to media coding and delivery, network protocols and quality of service, the work presented here focuses on interface architecture issues: how to structure the system and content to support such distributed interfaces for timed media applications?

By using physical interface devices, a more natural environment in which real-life stimuli for all the human senses are used, will give people more feeling of engagement [8]. Multimedia content can be distributed to several interface devices. For example, screens show the major part of the audio-visual content, surround audio equipments present the background music, ambient lights create harmonious atmosphere, and robotic toys render the expressions and actions of a character to react on the multimedia applications.

The carrier for this work is the development of an interactive storytelling application TOONS for children (age 8-12) in the context of NexTV project [15] sponsored by the European Commission under the IST-programme. In the TOONS show, the interactive content is distributed into the children's environment that involves several devices, i.e., a TV, a toy robot and a light. During the show, the children can interact with the content with the robot and the light. Figure 1 shows the conceptual model of TOONS. This model consists of storyline components and dialogue components. The storyline components comprise the non-interactive parts in the video stream. The dialogue components comprise the part in which the user can interact with objects in the stream to make choices. These components consist of a feed-forward and a feedback part and a decision point. In the middle of the sequence there are several decision points where the user can choose from different options in the story. Different choices at the decision points will lead to different storylines.

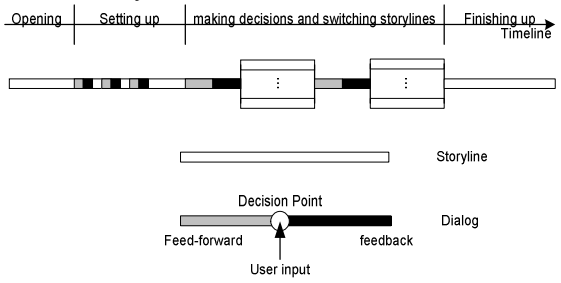

Figure 1. Conceptual model of TOONS

1.1. Requirements

According to the user requirements, the system architecture should emphasize and support the following:

1. Distributed Interfaces. Distributed interfaces mean that not only the content presentation, but also the user input and control are distributed over the networked environment. Different sets of input and output devices can constitute an environment.

2. Context Dependent Interaction. Here the term "context" means the environment configuration, the application context, and the user preferences. The target system platform can vary from a simple TV set with a set-top box, to a complicated home network environment. The configuration of such an environment is dynamic in both space and time dimensions. The user may activate or introduce new interface devices during the show. The application has to "know" what kind of environment it is running on at every moment and has to adjust itself to fit the current environment. The way of interaction may also depend on the application context. For example, in order to illuminate a dark room in the virtual world created by the application, a user can actually simply switch on a real light instead of pressing up or down buttons on a remote control. However, the user may still choose the remote control because he/she doesn't like to turn the light on, even though there is such a light available. The user, not the system, decides which way of interaction is preferred throughout the interactive show.

3. Synchronized Media and Interaction. In an interactive media application not only the media, but also the interactions are timed and should be synchronized with each other. Furthermore, this has to be done in an environment which consists of many interface devices. Multiple representations of the content or its parts should be distributed and synchronized on these devices according to their nature and the application semantics. A time dependent change-propagation mechanism should be developed for user-system interaction to ensure that all concerned system components are notified of changes to the content or the configuration, at the right moments in time.

1.2. Related work

The requirement for distributed interfaces challenges media documentation technologies. It requires that the documentation technology is able to deal with the distribution of the interaction and the media objects, in an environment which consists of multiple devices. The Binary Format for Scenes (BIFS) based MPEG-4 documentation [11,12] emphasizes the composition of media objects on one rendering device. It doesn't take the multiple interactors into account, nor does it have a notation for distributed interfaces. SMIL 2.0 [18] introduces the *MultiWindowLayout* module, which contains elements and attributes providing for creation and control of multiple top level windows. This is very promising and comes closer to the requirements of distributed content interaction. Although these top level windows are supposed to be on the same rendering device, they can to some extent be recognized as an interfacing component with the same capability.

As to the interface architectures for playing back systems, there exist many solutions for interactive media, although few of them take distributed interfaces into account. In [16], a typical user interface structure for digital TV applications is introduced, in which the graphical user interface (GUI) and the media content are clearly separated. Similar structures can be found in Immersive broadcasting [9]. An “immersive broadcast” application for sports events is presented in [14]. In this application, the consumer can compose his own personal show from a variety of streams of audiovisual and graphics data. Conceptually, video clips, text and graphics are overlaid on top of the TV program to provide a richer and more compelling experience for the viewer. However these structures have no possibilities whatsoever for content presentation on multiple networked devices and distributed interfaces.

Other than presenting the same media to different environments, many other projects focus more on dedicated environments and end user experiences. In the KidsRoom [2], images, music, narration, light and sound effects are used to transform a normal child’s bedroom into a fantasy land where children are guided through a reactive story. The LISTEN project [7] provides users with intuitive access to personalized and situated audio information spaces while they naturally explore the environments. In the DanceSpace, Networked Circus and TheaterSpace [16], a “media actors” software architecture is used in conjunction with real-time computer-vision based body tracking and gesture recognition techniques to choreograph digital media together with human performers or museum visitors.

In this paper we emphasize the issues of mapping the same media document onto different environments that have different configurations.

2. STORYML

To satisfy the requirements, the first question which has to be answered is how to describe such an interactive story that will be played in a distributed environment. We developed Story Markup Language (StoryML). StoryML is an XML based language for interactive stories that can describe how content can be served, received, and processed over the network and finally be played in different distributed environments.

2.1. Environment and interactors

The interactive story will be played in an environment which consists of several networked devices. We abstract these interactive units as Interactors. An Interactor is a self-contained entity which has an expertise of data processing and user interaction. Its input and output facilities form an interface with which a user can interact. It is able to abstract the user inputs as events and to communicate with other interactors. An Interactor can be present in an environment as a software entity, alive in a computer system or embodied in a hardware device.

An Environment is then defined as a dynamic configuration of many of these Interactors. The Environment assigns different tasks to each of the Interactors according to the definition of a story script, for example, rendering media objects, reporting the user responses during different periods of time.

2.2. Media Objects

Storylines, feed-forward and feedback components are all timed media objects. A timed media object is defined as a timed data stream which can be rendered by any of the interactors in the environment, and can be perceived by a user via any or many channels of perception.

As its definition implies, a media object can be rather abstract, for example, expressions, behaviors, and even emotions, can be defined as a media object as long as it can be recognized and rendered by any Interactors. The abstraction of media objects provides possibilities for the content producer to describe a story at a high level without knowing the details of the environment configuration, e.g., a content producer can specify a robot to show the 'happiness' behavior without the need to know whether there is a robot present in the Environment or not, and if there is one present, how this robot will show the 'happiness' behavior. It solely depends on the configuration of the Environment and the implementation of the robot.

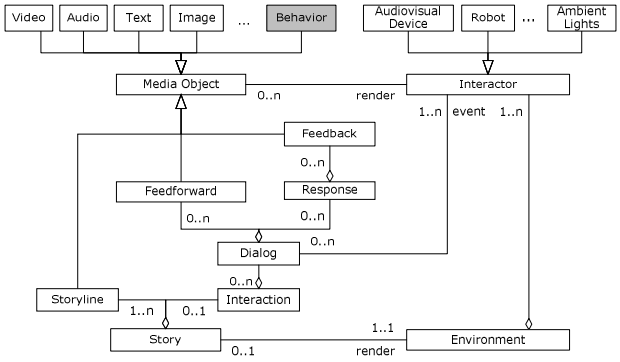

Figure 2. StoryML object model

The StoryML reflects directly many concepts from the object-oriented model (Figure 2). The major objective of doing so is to make the StoryML an easy authoring language for content producers. It provides a higher level of abstraction which is independent of media representation technologies. The detailed definition and a example script can be found in [10].

3. STORYML PLAYER

StoryML has been defined as a solution for interactive story documentation, in which the distributed presentation environment has been taken into account. Now the task is to design appropriate software architecture for the StoryML player. We choose the PAC based architecture.

3.1. PAC or MVC?

Many interface architectures have been developed along the lines of the object-oriented and the event processing paradigms. Model-View-Controller (MVC) and Presentation-Abstraction-Control (PAC) are the most popular and often used ones [4].

The MVC model is an agent-based architecture. It divides an agent into three components: model, view and controller, which respectively denotes processing, output and input. The model component encapsulates core data and functionality. View components display information to the user. A View obtains the data from the model. There can be multiple views of the model. Each view has an associated controller component. Controllers receive input, usually as events that encode hardware signals from a keyboard, a mouse or a remote control.

The interactive application is modeled as a set of PAC agents whose communication scheme forms a hierarchy [5] (Figure 3). The PAC based architecture is to be shown in Figure 5, an agent has a presentation component for its perceivable input and output behavior, an abstraction for its function core, and a control to express dependencies. The control of an agent is in charge of the communication with other agents and of expressing dependencies between the abstract and the presentation components of the agent. In PAC, the abstraction and presentation components of the agents are not authorized to directly communicate with each other or with their counterparts of other agents. Dependencies of any sort are conveyed via the controls of the agents.

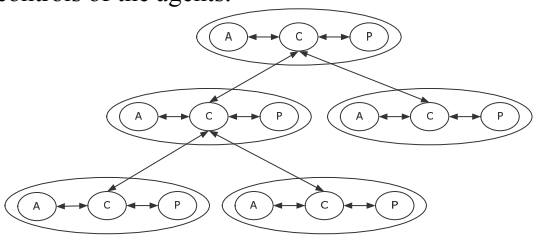

Figure 3.Hierarchy of PAC agents [5]

We consider The PAC based architecture more suitable for the StoryML player than MVC, because of the following reasons:

1. StoryML involves independent devices as physical interactors. It should have the ability to adapt to the changing configuration. PAC can satisfy these requirements by separating self-reliant subtasks of a system into cooperating but loosely-coupled agents. Individual PAC agents provide their own human-computer interaction. This allows the development of a dedicated data model and user interface for each semantic concept or task within the system. PAC agents can be distributed easily to different threads, processes or machines.
2. The PAC based architecture emphasizes the communication and cooperation between agents with a

mediating control component. It is crucial to have such a mechanism for a distributed application like the StoryML player. In the PAC architecture, all agents communicate with each other via their control component with a pre-defined interface. So, existing agents can dynamically register new PAC agents to the system to ensure communication and cooperation.

3. The input and output channel of the individual interactors in StoryML are often coupled. In the MVC architecture, controller and view are separate but closely-related components, whereas the PAC architecture takes this intimate relationship between these two components into account and considers the user accessible part of the system as one presentation component.

4. The StoryML player has to facilitate content based interaction, which means that the user can interact with interactive media objects in the content. The media objects and the attached possible interaction are often documented together as an entity, which will be rendered by one of the interactors. At a conceptual level, this request can be easily assigned to the presentation component of the interactor. Separating the attached operation from the media object will increase the complexity.

3.2. Extending PAC for timed media

The overhead in the communication between PAC agents may impact the efficiency of the system. For example, if a bottom-level agent retrieves data from the top-level agent, all the intermediate-level agents along the path from the bottom to the top of the PAC hierarchy are involved in this data transportation. If agents are distributed, data transfer also requires inter-process communication, together with marshaling, un-marshaling, fragmentation and re-assembling of data [4].

To overcome this potential pitfall, the StoryML player extends the abstraction component. For timed media, each abstraction component is also considered as a media processor, which takes a MediaSource as input, performs some processing on the media data, and then outputs the processed media data. It can send the output data to a presentation component or to its MediaSink (Figure 4).

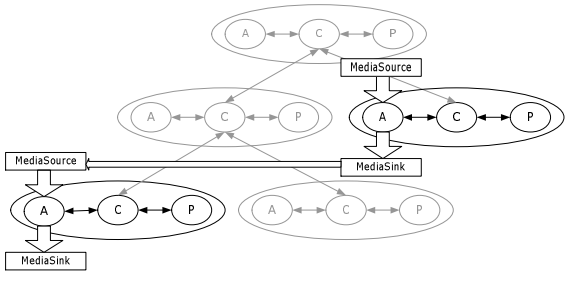

Figure 4. PAC for timed media

Regarding the PAC hierarchy as a network, an agent with a MediaSink attached to its abstraction component can be viewed as a streaming media server and those agents which require a MediaSource can be viewed as streaming media clients. A direct pipeline can be built between a MediaSink and a MediaSource and the media can be streamed through the pipeline with real-time streaming protocols. Pipelines can be built and cut off only by the control components. Thus, the control hierarchy remains intact.

3.3. Architecture of the StoryML Player

Figure 5 shows the hierarchical structure of the StoryML player. The content portal establishes the connection to content servers and provides the system with timed content. The content pre-fetcher overcomes the start latency by pre-fetching a certain amount of data and ensures that the media objects are prepared to start at specified moments.

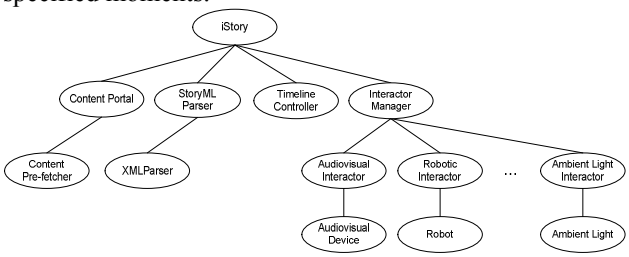

Figure 5. PAC based architecture of the StoryML player

The interactive stories are documented in StoryML. An XML parser first parses the document into Document Object Model (DOM) objects and then the StoryML parser translates the DOM objects into internal representations. The StoryML player also maintains a timeline controller, which plays an important role in synchronizing user interaction with the story.

The bottom level agents indicate different physical interface devices. These physical devices are often equipped with embedded processors, memory, and possibly with some input and output accessories. More physical agents can be involved into the architecture at this level.

For each physical agent, there is an intermediate virtual interactor connected as its software counterpart. Provided with this layer of virtual interactor, the system can achieve the following:

1. Decoupling of media processing from the physical interface devices and enabling process distribution. It is possible to assign media processing tasks of a physical agent, such as decoding a stream or composing a scene, to another more capable device in the network, by moving the virtual interactor to that device. The result of the processing is then transferred back to the physical presentation component of the physical agent for direct rendering. The media processing, therefore, can also be distributed to the network.

2. Easy switching of the user interaction from the physical device to its virtual counterpart or vice versa. The virtual interactors maintain the configuration of the system to observe and verify the availability of interface devices. If the environment can not satisfy the story with the preferred interface devices, the system can always provide alternatives. If a physical device is not available in the environment or the user prefers interacting with the virtual interactors, then the virtual interactor functions as the substitute and presents its interface on a screen which is manipulated by the user with standard input devices such as a keyboard or a mouse.

3. Satisfying the requirements for the variety of the interface devices. These virtual interactors can be viewed as software drivers for physical interactors, which hide the differences between diverse yet homogeneous devices, and provide the higher level agents with the same interface.

4. The virtual interactors are coordinated by an interactor manager. The interactor manager creates software interactors and transfers user-events between software interactors and keeps them synchronized.

3.4. Media and Interaction Synchronization

One of the key characteristics of the StoryML system is related to media and interaction synchronization issues. In this section the system is classified and described according to the synchronization reference model, presented in [1].

3.4.1. Specification layer

In StoryML, media objects and interaction dialogues refer to an implicit timeline by specifying their starting and stopping point in time. The metaphor behind it can be easily understood by comparing with the conceptual model of the interactive story. Synchronizing objects by means of a timeline allows a very good abstraction from the internal structure of single-medium objects and composite multimedia objects. Define the beginning of a video presentation to an audiovisual interactor in a story requires no knowledge of the related video frames. The timeline approach is therefore quite intuitive and easy to use in authoring situations.

3.4.2. Object layer

A crucial part of the StoryML system is dedicated to provide object layer services. Input StoryML documents are analyzed using an XML parser in order to build a structural object representation of the synchronization specification. The object structure mirrors the StoryML document structure to a schedule for the presentation, which is managed by a timeline controller. The StoryML system implements a global timeline controller (Figure 5) for synchronizing media objects and interaction, which might be distributed to several interactors.

The presentation of a document is managed at object layer through close communication with stream layer entities. Scheduling constraints are mapped to stream layer method invocations, which control the playback and synchronization of media streams.

In StoryML, possible user interactions are authored with defining dialogs. These dialogs are registered to the timeline controller. At a predefined moment, the timeline controller initializes a dialog with starting several media objects on target interactors, as feed-forward information. The dialog then requests the interactors to listen to the user events. The StoryML doesn't associate any user input to a specific media object, but an interactor instead. If the user reacts, the interactor will abstract the user response as an event and this event will trigger feedback media objects. If the user event results in a later change, then register this change to the timeline controller. Thus, the user interaction is synchronized with the media presentation.

3.4.3. Stream layer

In the StoryML system, each media object implements a MediaClock to keep track of time for a particular media object. The MediaClock defines the basic timing and synchronization operations that are needed to control the presentation of media data. Even time-independent media objects, such as image, test, graphics and robotic behaviors, are attached with a MediaClock. Therefore, all the media objects can be viewed as time-based media, or media streams.

4. TOONS IMPLEMENTATION

For demonstration purpose, the TOONS application is implemented on a PC, a robot and a light, which respectively serve as an audiovisual interactor, a robotic interactor and an ambient light interactor. All these components are implemented based on the Java technology. The PC provides services of the content portal, StoryML parser, timeline controller and interactor manager. The robot, named Tony (Figure 6), is assembled using theLEGO Mindstorms Robotics Invention System (RIS) [17]. TONY is powered by LeJOS, an embedded Java VM [13]. The light is controlled via a Java virtual interactor running on the PC.

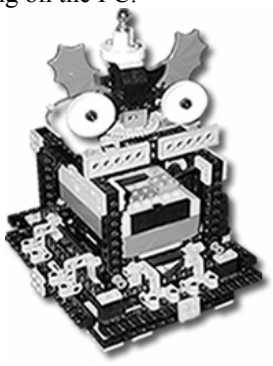

Figure 6. Tony

In the TOONS show these interactors are involved. Tony will be ‘woken up’ at a certain moment during the show. Tony is then able to reacts on some events happening in

the show. The user will be invited to help the main character in the show to make decisions (for example to open the "left" or the "right" door) during a certain periods in time by playing with Tony. When the user decides the character to enter the dark room, the light will be off and Tony behaves scared. The user can then switch on the light to illuminate both the dark room in the reality and the dark room in the virtual world.

We demonstrate the adaptive architecture by adding and removing the light and Tony from the environment during the real-time show.

5. CONCLUSIONS

We presented a framework which allows the presentation of multimedia contents in different Environments. Each Environment can consist of several interactors being distributed over a network. StoryML plays an important role in the framework:

1. StoryML allows the separation of the content from the concrete physical devices

2. A StoryML document specifies abstract media objects at a high level and leaves the complexity to the implementation of the rendering interactors.

3. StoryML supports the automatic mapping of the same document to different Environments, or a dynamic Environment.

In the implementation of the StoryML system, Media objects are distributed to interactors, and synchronized with a timeline controller. Centralized synchronization requires a stable and fast network infrastructure to ensure that timed events can reach the interactors in time.

An implicit assumption has been made in the design and implementation of the StoryML system: In the user's Environment, there is at least an audiovisual interactor with a screen and input accessories, on which the virtual software interactors can always present themselves if their physical counterpart is not available. This limits the use of StoryML framework for an interactive show which does not require any visual presentation, e.g. an interactive radio show.

REFERENCES

[1] Blakowski, G. & Steinmetz, R. (1996). A Media Synchronization Survey: Reference Model, Specication, and Case Studies. IEEE Journal on Selected Areas in Communications, 14(1), 5-35.

[2] Bobick, A., Intille, S., Davis, J., Baird, F., Pinhanez, C., Campbell, L., Ivanov, Y., Schuttle, A. & Wilson, A. (1999). The KidsRoom: A Perceptually-Based Interactive and Immersive Story Environment. Presence: Teleoperators and Virtual Environments, 8(4), 367-391.

[3] Buford, J. F. K. (1994). Architectures and Issues for Distributed Multimedia Systems. In J. F. K. Buford (Ed.), Multimedia Systems (pp. 45-46). Addison Wesley.

[4] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. & Stal, M. Pattern-Oriented Software Architecture, A System of Patterns. Chichester, UK: John Wiley & Sons, Inc., 1996.

[5] Calvary, G. et al. From single -user architectural design to PAC*: a generic software architecture model for CSCW. CHI'97 Conference 1997), Addison Wesley, 242-249.

[6] Duke, D.J..&.Herman, I. A Standard for Multimedia Middleware. the 6th ACM International Conference on Multimedia '98 (Bristol, UK, 1998), 381-390.

[7] Eckel, G. The Vision of the LISTEN Project. the Seventh International Conference on Virtual Systems and Multimedia (VSMM'01) 2001), IEEE,

[8] Eric, S. (2000). Interactive Toy characters as Interfaces for Children. In anonymous (Ed.), Information Appliances and Beyond: Interactive design for consumer products Morgan Kaufmann Publishers.

[9] Herrmann, L. (2000). Immersive Broadcast: Concept and Implementation. (Rep 748). Philips Research LEP.

[10] Hu, J. (2001). Distributed Interfaces for a Time-based Media Application. Post-master thesis, Eindhoven University of Technology, Eindhoven.

[11] Koenen, R. (1999). MPEG-4: Multimedia for Our Time. IEEE Spectrum, 36(2), 26-33.

[12] Koenen, R. Overview of the MPEG-4 Standard. http://mpeg.telecomitalialab.com/standards/mpeg-4/mpeg-4.htm.

[13] LeJOS LeJOS: Java for the RCX. Retrieved October 02, 2002, http://www.lejos.org.

[14] Mallart, R. (1999). Immersive Broadcast Reference Application. (White Paper). Philips Research.

[15] NexTV The NexTV Project Homepage. http://www.extra.research.philips.com/euprojects/nextv.

[16] Peng, C..&.Vuorimaa, P. Development of Java User Interface for Digital Television. 8th International Conferences in Central Europe on Computer Graphics, Visualization and Computer Vision 2001),

[17] The LEGO Group The LEGO Mindstorms Homepage. http://mindstorms.lego.com/.

[18] W3C Synchronized Multimedia Integration Language (SMIL 2.0). http://www.w3.org/TR/smil20/cover.html.