Proceedings of the 21st IASTED International Conference
APPLIED INFORMATICS
February 10-13, 2003, Innsbruck, Austria

AN ADAPTIVE ARCHITECTURE FOR PRESENTING INTERACTIVE MEDIA ONTO DISTRIBUTED INTERFACES

Jun Hu
Dept. of Industrial Design
Eindhoven University of Technology
Den Dolech 2, 5600MB Eindhoven, The Netherlands
Media Interaction Group
Philips Research Laboratories
Prof. Holstlaan 4, 5656AA Eindhoven, The Netherlands

Loe Feijs
Dept. of Industrial Design
Eindhoven University of Technology
Den Dolech 2, 5600MB Eindhoven, The Netherlands

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 [1,2]. 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 [3]. Multimedia content can be distributed to several interface devices. For example, screens show the major part of the audiovisual content, surround audio equipment presents the background music, ambient lights create harmonious atmosphere, and robotic toys show the expressions and actions as a character.

The carrier for this work is an interactive storytelling application TOONS for children (age 8-12) in the NexTV project 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, using the robot and the light. Figure 1 shows the TOONS conceptual model. This model consists of storyline components and dialog components. The storyline components comprise the non-interactive parts in the video stream, whereas the dialog components comprise the interactive parts with which the user can make choices. A dialog consists of a feed-forward, a feedback part and a decision point. Different choices at the decision points will lead to different storylines.

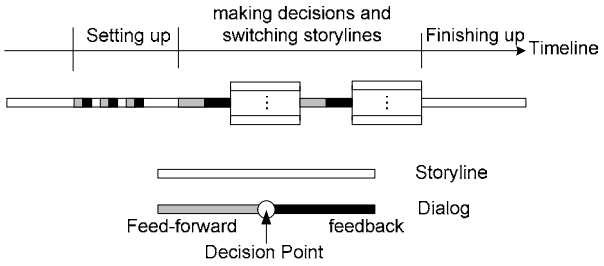

Figure 1. TOONS conceptual model

1.1 Requirements

According to the user requirements, the system architecture should 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 devices. Different sets of input and output devices can form a different 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 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 environment it is running on at every moment and has to adjust to the current environment. The way of interaction may also depend on the application context. For example, to illuminate a dark room in the virtual world created by the application, a user can simply switch on a real light instead of pressing buttons on a remote control. However, the user may still choose the remote control because she doesn't like to turn the light on, even though there is such a light available.

3. Synchronized Media and Interaction. In an interactive media application not only the media, but also the interactions are timed and should be synchronized on Multiple devices according to their nature and the application semantics. A time dependent change-propagation mechanism is to be developed to notify the user events and the content changes to all concerned system components, at the right moments in time.

1.2 Related work

The requirement for distributed interfaces challenges media documentation technologies. It needs the documentation technology to be able to describe the user interaction and the media presentation with multiple devices. The Binary Format for Scenes (BIFS) based MPEG-4 documentation [4] stresses 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 [5] introduces the *MultiWindowLayout* module, which contains elements and attributes for creation and control of multiple top level windows. This is promising and comes closer to the requirement of distributed interfaces. 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 of 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. [6] introduces a typical user interface structure for digital TV applications, in which the graphical user interface (GUI) and the media content are clearly separated. A similar structure is found in Immersive Broadcasting [7]. An "immersive broadcast" application for sports events is presented in [8]. In this application, the consumer can compose his personal show from various 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 provide no means for content presentation on multiple networked devices.

Other than presenting the same media to different environments, many other projects focus more on dedicated environments and end user experiences. In the KidsRoom [9], 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 [10] 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 [11], 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 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 to be answered is how to describe such an interactive story that can be played on multiple devices. We developed Story Markup Language (StoryML), 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

Multiple interactive devices will show the story in an environment. We abstract these interactive units as Interactors. An Interactor is a self-contained entity which has an expertise of data processing and user interaction. 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 setup of multiple Interactors. The Environment assigns different tasks to each of the Interactors according to 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 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 channels of perception.

As its definition implies, a media object can be rather abstract, for example, expressions, behaviors, and even emotions, can be 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 every environment. For instance, a content producer should be able to specify a robot to show a specific behavior without the need to know whether there is a robot present in the Environment, and if there is one present, how exactly the robot shows the behavior.

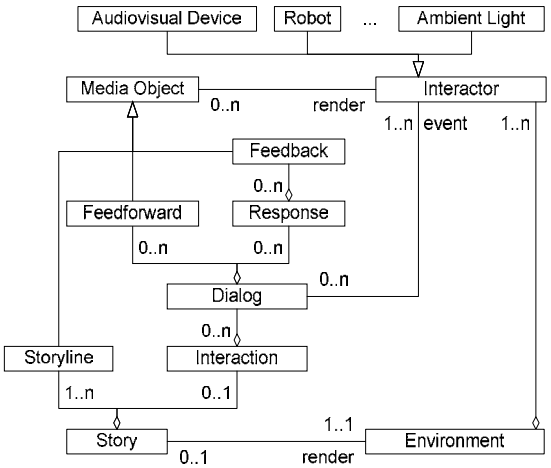

Figure 2. StoryML object model

The StoryML reflects directly many ideas 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 abstraction which is independent of media representation technologies. The detailed definition and an example script are in [12].

3 STORYML PLAYER

We have defined the StoryML as a solution for interactive story documentation, taking the distributed interfaces into account. Now the task is to design a proper software architecture for the StoryML player. We choose the PAC [13] 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 [14].

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.

In a PAC based architecture, a set of PAC agents forms a hierarchy by the communication scheme [13] (Figure 3). 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 communicate directly with one another or with their counterparts of other agents. Dependencies of any sort are conveyed by the controls of the agents.

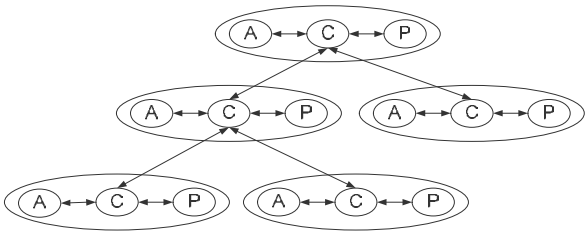

Figure 3.Hierarchy of PAC agents [13]

We consider The PAC based architecture to be 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 capacity to adapt to the changing configuration. PAC can satisfy the need 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 to develop 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 one another via their control components with a predefined communication interface. So, existing agents can dynamically register new PAC agents to the system to ensure communication and cooperation.
3. The input and output channels of the individual interactors in StoryML are often coupled. In an 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. A media object and the attached possible interaction are often documented together as an entity, which is to 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, transferring data from a top-level agent to a bottom-level agent involves all the intermediate-level agents along the path. If agents are distributed, data transfer also requires inter-process communication, together with marshaling, un-marshaling, fragmentation and reassembling of data [14].

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).

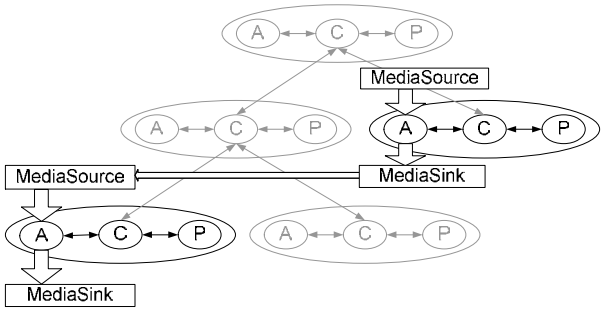

Figure 4. PAC for timed media

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*. 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 sets up 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.

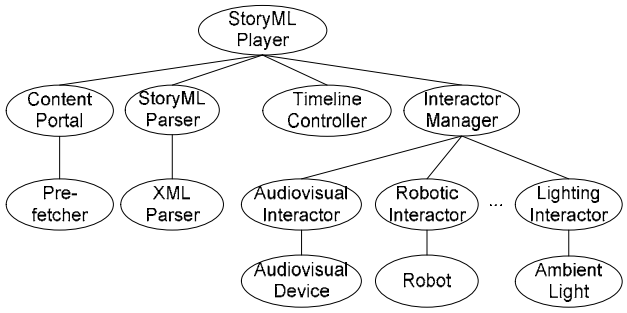

Figure 5. PAC based architecture of the StoryML player

An XML parser first parses the StoryML 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 synchronization.

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. An arbitrary number of physical agents can be added to 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 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. The user may then use standard input devices such as a keyboard or a mouse for interaction.

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 interactor manager coordinates the virtual interactors by creating software interactors and transferring 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 about media and interaction synchronization issues. In this section we classify and describe the synchronization issues of the system using the synchronization reference model, presented in [15].

3.4.1 Specification layer

In StoryML, media objects and interaction dialogs refer to an implicit timeline by specifying their starting and stopping time. The metaphor behind it can be easily understood by comparing to the conceptual model of the interactive story. Synchronizing objects with a timeline allows a 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 needs no knowledge of the related video frames. The timeline approach is therefore intuitive and easy to use in authoring situations.

3.4.2 Object layer

An important part of the StoryML system is dedicated to provide object layer services. An XML parser analyzes the input StoryML documents to build a structural object representation of the synchronization specification. The object structure mirrors the StoryML document structure to a schedule for the presentation, managed by a timeline controller. The StoryML system implements a global timeline controller (Figure 5) to synchronize the media objects and the interaction that are distributed to several interactors.

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

In StoryML, dialogs define possible user interactions. The timeline controller registers these dialogs. 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. 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 the time controller registers the change for later triggering. Thus, the user interaction is synchronized with the media presentation.

3.4.3 Stream layer

The StoryML player attaches a *MediaClock* to each media object to keep track of the media time. The *MediaClock* defines the basic timing and synchronization operations that are needed to control the timed media presentation.

4 TOONS IMPLEMENTATION

For demonstration purpose, the TOONS application is implemented on a PC, a robot and a light, which respectively serves as an audiovisual interactor, a robotic interactor and an ambient light interactor. All these components are implemented using the Java technology. The PC provides the services of the content portal, the StoryML parser, the timeline controller and the interactor manager. The robot, named Tony (Figure 6), is assembled using the LEGO Mindstorms Robotics Invention System (RIS) [16]. Tony is powered by LeJOS, an embedded Java VM [17]. The light is controlled via a Java virtual interactor running on the PC.

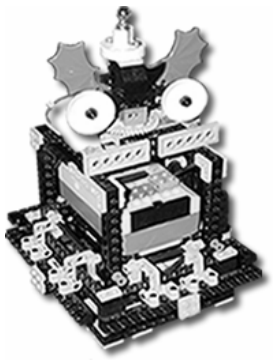

Figure 6. Tony

In the TOONS show, after being ‘woken up’ at a certain moment, Tony is able to react on some events happening in the story. The user is 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 to let the character go into the dark room, the light will be off and Tony behaves scared. The user can then switch on the light to light up 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 multiple interactors distributed over a network. StoryML plays an important role in the framework:

1. StoryML allows to separate 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 needs 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.

6 ACKNOWLEDGEMENTS

The research reported in this paper is done in the context of NexTV project that is partly funded by the European Commission under the IST-programme.

We are grateful to those who read and commented on the early drafts, and in particular to Emile Aarts, Maddy Janse and Warner ten Kate.

REFERENCES

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

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

[3] E. Strommen, Interactive Toy characters as Interfaces for Children, in E. Bergman(Ed.), *Information Appliances and Beyond: Interactive design for consumer products* (New York: Morgan Kaufmann Publishers, 2000), 257-298.

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

[5] M. Slowinski, T. Kennedy et al, *SMIL: Adding Multimedia to the Web* (Sams, 2001).

[6] C. Peng and P. Vuorimaa, Development of Java User Interface for Digital Television, *the 8th International Conference on Computer Graphics, Visualization and Interactive Digital Media*, Czech Republic, 2000, 120-125.

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

[8] R. Mallart, Immersive Broadcast Reference Application, White Paper, Philips Research, 1999.

[9] A. Bobick, S. Intille, J. Davis et al, The KidsRoom: A Perceptually-Based Interactive and Immersive Story Environment, *Presence: Teleoperators and Virtual Environments, 8*(4), 1999, 367-391.

[10] G. Eckel, Immersive Audio-Augmented Environments - The LISTEN Project, *the 5th International Conference on Information Visualization (IV2001)*, Los Alamitos, CA, USA, 2001,

[11] F. Sparacino, G. Davenport and A. Pentland, Media in performance: Interactive spaces for dance, theater, circus, and museum exhibits, *IBM Systems Journal, 39*(3/4), 2000, 479-510.

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

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

[14] F. Buschmann, R. Meunier et al, *Pattern-Oriented Software Architecture, A System of Patterns* (Chichester, UK: John Wiley & Sons, Inc., 1996).

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

[16] M. Ferrari and G. Ferrari, *Building and Programming LEGO Mindstorms Robots Kit* (Syngress Media Inc, 2002).

[17] B. Bagnall, *Core LEGO MINDSTORMS Programming: Unleash the Power of the Java Platform* (Prentice Hall PTR, 2002).