StoryML: Towards Distributed Interfaces for Timed Media

Jun Hu

Media Interaction Group
Philips Research, Eindhoven
hu@natlab.research.philips.com

Department of Industrial Design
Eindhoven University of Technology
j.hu@tue.nl

ABSTRACT
This paper addresses the architectural issues of presenting interactive timed media to distributed and networked devices. An experimental system was developed in which the interactive content is documented in StoryML, an XML-based language, and presented to multiple interface devices organized in an agent-based architecture. The StoryML allows the separation of the content from concrete physical devices, the definition of abstract media objects and the automatic mapping of the same document to different environments. It can serve as a basic framework for presenting interactive timed media in a networked environment.

Keywords
Distributed interfaces, timed media, interactive content, content documentation

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 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 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 (named TOONS) for children (age 8-12) in the context of NexTV project [4]. A user-centered methodology was used for the fine-tuning of the requirements. Based on these refined requirements, a conceptual design of the interactive story was developed as input for the system design, and current technologies and existing architectures were also evaluated. The design and implementation of the system followed the Object-oriented methodology. The system was implemented using a specific set of technologies, i.e., eXtensible Markup Language (XML) and Java based technologies.

2 REQUIREMENTS

The story and interaction scenario were created with the cooperation of the end-users. Their input was used to create the interactive story and to develop conceptual model of the story. To collect this input user trials were conducted [5]. From raw video assets and 2D hand drawings a Macromedia Director movie was created. The interaction was limited to a few simple activities.

2.1 Conceptual model of TOONS

Figure 1 shows the conceptual model of TOONS. This model consists of storylines and dialogs. The storylines comprise the non-interactive parts in the video stream. The dialogs comprise the part in which the user can interact with objects in the stream to make decisions. A dialog consists of a feed-forward and a feedback part and a decision point. The application starts with an introduction or opening sequence, followed by a set-up sequence in which the user can choose different appearances for the main character in the story. In the middle of the sequence there are several decision points where the user can choose from different options in the story. For example, to open one of the presented different doors, the user can knock on different buttons on a console. Different decisions at the decision points will lead to several different storylines. The story will end up with a finishing sequence in which the scenario promises that more episodes are coming or that the whole story has just finished.

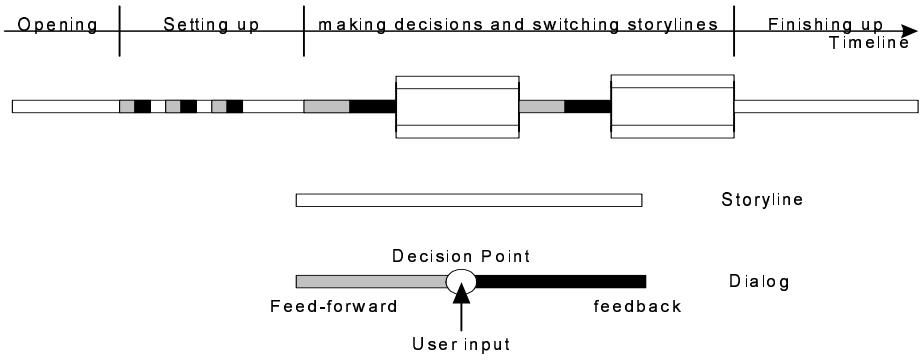

Figure 1. Conceptual model of TOONS

2.2 Technical requirements for the system architecture

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

2.2.1 Distributed interfaces

In the TOONS application, several different components can be distinguished from the user's perspective. Full screen audiovisual scenes entertain the users with the story. Interactive objects are present in the scenes, which can listen and react to the user input to personalize their storylines. Graphic user interfaces can be present as overlays on top of the scene, which can be menus, buttons, icons and arbitrary-shaped video clips, or a combination of them.

Different input and output devices can be used to interact with the content. Simple selections and choices can be made with a remote control, while mass data input and complex GUI operations can be done with a remote keyboard and a mouse. A smart card can be used to identify user profiles and to feed the application with predefined configurations. The LED display on the front panel of a set-top box can present extra text information with regard to the real-time streamed content.

The application can also play part of the content and get user responses from some networked devices in the home environment. These devices could be as simple as a bi-directional interface device that can play feed-forward and feedback information that is given by the application, e.g., an interactive toy with touch sensors and sound output. They can also be as sophisticated as robots that have their own behaviors and intelligence. Furthermore, a second screen may be used to present more extra media information; for off-line configuration and entertainment, a PC system can be connected.

Not only the designers, but also the users, have such an idea to introduce tangible interface devices into the TOONS application. During the user trials [5], Karolina, a 12-year old girl, suggested a robot as the tangible interface device (cf. Figure 2). "*As I understand, there will be a special device sold together with the program that can be used to make choice, right?*" She suggested some buttons on its hands as the input channel, and a touch screen in its "belly" as the output channel.

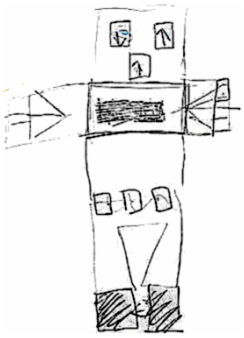

Figure 2. Karolina's robot

2.2.2 Context dependent interaction

Here the term "context" means the environment configuration, 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 program. The application has to "know" what kind of environment it is running on at every moment and adjust itself to fit the environment on the fly.

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 program.

2.2.3 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, 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 is needed for the 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.

2.3 When technologies meet the requirements

2.3.1 Media application documentation

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 interactors. The Binary Format for Scenes (BIFS) based MPEG-4[6] documentation 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 introduces the *MultiWindowLayout* module, which contains elements and attributes providing for creation and control of multiple top level windows [7]. 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 software interface components which have the same capability.

The TOONS application intends to make use of multiple interface devices with different capabilities, i.e., an audiovisual device and a tangible interface device. In this sense, neither MPEG-4 nor SMIL can fully satisfy the requirements.

2.3.2 Interface architectures

1. Future TV

In [8], a typical user interface structure for digital TV applications is introduced. It includes not only graphics but also timed media. Figure 3 illustrates this basic user interface structure for interactive television services or applications on a set-top box. The user interface is composed of graphical user interface (GUI) and broadcasting content. The GUI includes graphics and user input as the so called Look & Feel.

In this structure, however, the GUI is not part of the content. The user may select different content by manipulating GUI widgets. This interaction does not enable direct manipulation of anything inside the content. The content producer and service provider serve the content and content-related information. How the interface should look like and how the user can operate it depend on the implementation of the local platform.

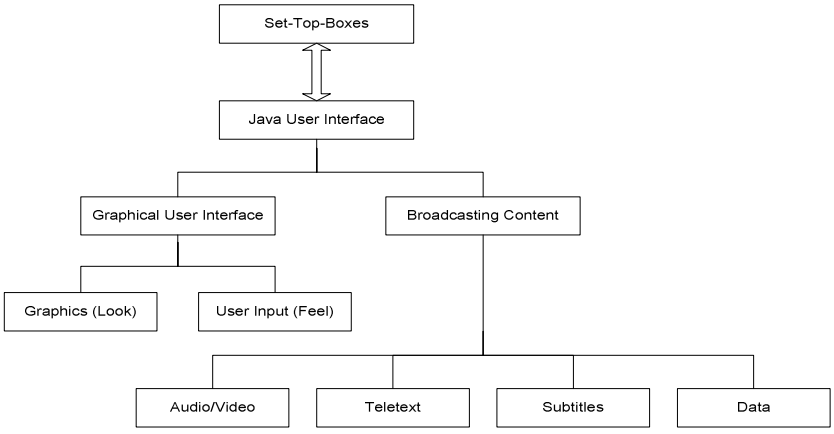

Figure 3. Structure of the user interface in digital TV [8]

This structure has no possibilities whatsoever for content presentation on networked multiple devices and distributed interfaces, nor for synchronization between the media and the interaction.

2. Immersive broadcast

Immersive broadcast is a generic term for interactive multimedia applications mainly targeting enhanced digital television programs [9]. An "immersive broadcast" application for sports events is presented in [10]. In this application, the consumer can compose his own personal program 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.

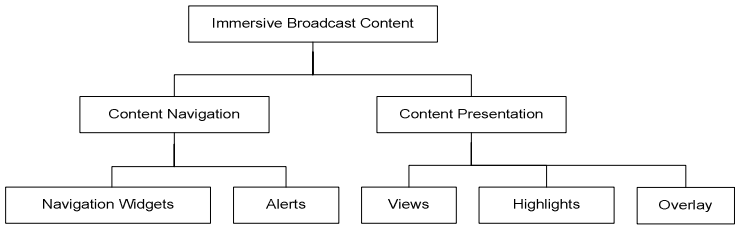

Figure 4. "Immersive broadcast" components

The components of an "immersive broadcast" application can be categorized into content presentation components (views, highlights and the overlay) and content navigation components (cf. Figure 4). Compared to the structure shown in Figure 3, here the user interface components are a part of the broadcasting content. The user interaction will influence the presentation of the live or stored content, or content related information.

This structure doesn't take into account the distributed presentation and interaction either.The user interaction remains at the level of content navigation, which can not change directly what is in the content.

In short, the current technologies described above can hardly satisfy the requirements for the distributed media presentation and interaction. For this we need a different approach.

3 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. The existing open standards can not take more than one multimedia terminal into account. They don't describe an environment in which multiple distributed interfaces are incorporated instead of only one multimedia terminal. To account for this problem, a description language was developed. This language, Story Markup Language (StoryML), is an XML based specification language for interactive stories which aims at a technical solution for the interactive story representation. It can describe how content can be served, received, and processed over the network and finally played in a distributed environment.

3.1 Environment and interactors

The interactive story will be played in an environment which consists of many networked devices, such as audiovisual screens, surrounding audios systems, ambient lights, and robotic toys. These devices are abstracted 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 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.

An implementation of the TOONS application presented later in this paper requires that an Environment should have at least two physical Interactors: an audiovisual screen and a robotic toy. When the robotic toy doesn't exist in the Environment, a software agent is implemented as an alternative for its hardware counterpart.

3.2 Media Objects

Storylines, feed-forward and feedback components are all timed media objects. A timed media object is defined as a 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.

Storylines, feed-forward and feedback components are all timed media objects. A timed media object is defined as a data stream which can be rendered by any of the Interactors in the Environment. They can be perceived by users in different ways. As its definition implies, a timed 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 an Interactor. In [11], it was shown that abstract expressions representing emotions could be as convincing as natural human faces to convey a meaning to users. 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.

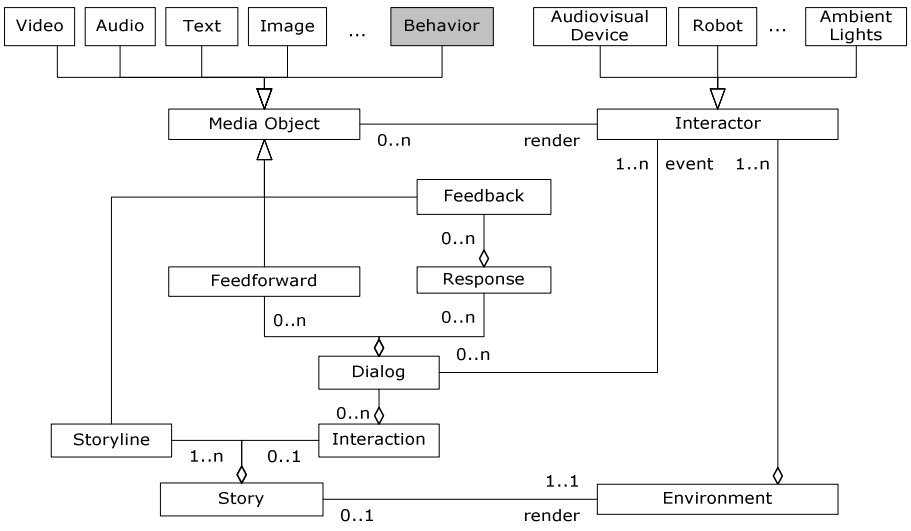

Figure 5. StoryML object model

StoryML reflects directly many concepts from the object-oriented model (cf. Figure 5). The major objective of doing so is to make 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 can be found in [12] or [13].

3.3 An Example

The following StoryML document (Figure 6) represents a TV show in which two Interactors are involved, i.e., an audiovisual screen and a toy robot Tony. Tony will be 'woken up' 25 seconds after the show has started. The user will be invited to help the main character in the show to make a decision (for example "left" or "right") at a certain period of time by playing with Tony.

```
<?xml version="1.0"?>
<!DOCTYPE StoryML SYSTEM "StoryML.dtd">

<StoryML>
   <environment id="ToonsPlatform">
      <interactor id="screen" type="audiovisual" />
      <interactor id="Tony"   type="robot" />
   </environment>

   <story id="TOONS" title="TOONS (c) Philips Research" >
        <storyline id="HappyGarden"
                 src="file:E:/happygarden.mpg" interactor="screen" />
        <storyline id="AngryGarden"
                 src="file:E:/angrygarden.mpg" interactor="screen" />

        <interaction>
               <dialog id="WakeupTony" begin="25000" end="65000" >
                     <feedforward content="relaxed"
                              type="behavior" interactor="Tony" />
               </dialog>
               <dialog id="WhichDoorToEnter"
                       begin="37000" end="52000" wait="48000" >
                       <feedforward content="attention"
                            type="behavior" interactor="Tony" />
                       <response interactor="Tony" event="left" default="yes"
                            storyline="HappyGarden" action="switchto" >
                              <feedback src="file:E:/door_open_blue_feedback.mpg"
                                   type="video" interactor="screen" />
                       </response>
                       <response interactor="Tony" event="right"
                       storyline="AngryGarden" action="switchto">
                              <feedback src="file:E:/door_open_green_feedback.mpg"
                                   type="video" interactor="screen" />
                       </response>
               </dialog>
        </interaction>
   </story>
</StoryML>
```

3.4 Timeline

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.

4 IMPLEMENTING THE 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 an

appropriate software architecture for the StoryML player. We choose the PAC based architecture.

4.1 PAC or MVC?

Many interface architectures have been developed along the lines of the object-oriented and the event processing paradigms. MVC (Model-View-Controller) and PAC (Presentation-Abstraction-Control) 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 the PAC architecture, 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. The interactive application is modeled as a set of PAC agents whose communication scheme forms a hierarchy [15] (cf. Figure 6).

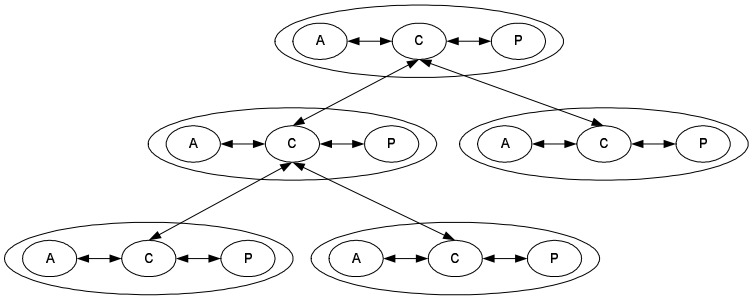

Figure 6. Hierarchy of PAC agents [15]

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

- 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.

- 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.

- 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.

- 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 operation 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 this interactor. Separating the attached operation from the media object will increase the complexity.

4.2 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 [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 7).

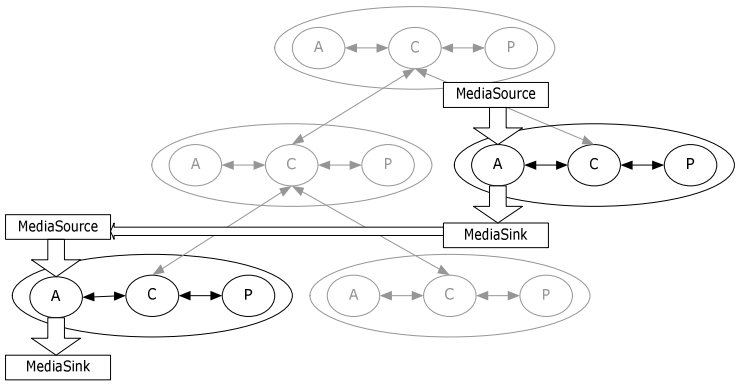

Figure 7. PAC for timed media

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.

4.3 Architecure of the StoryML Player

Figure 8 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.

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 bottom level agents indicate different physical interface devices while residing in these 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.

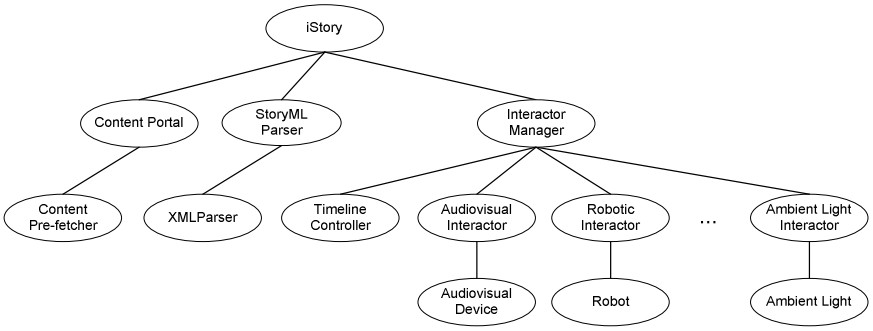

Figure 8. PAC based architecture of the StoryML player

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:

- 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.

- 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.

- Satisfying the requirements for the variety of the interface devices. These virtual interactors can be viewed as software drivers for physical agents, which hide the differences

between diverse yet homogeneous devices, and provide the higher level agents with the same interface.

The software interactors are coordinated by an interactor manager. The presentation component of the interactor manager provides the interface for creating and ending different interactions and navigations between these software interactors. The interactor manager transfers user-events between software interactors and keeps them synchronized. The interactor manager also maintains a timeline controller, which plays an important role in synchronizing user interaction with the story.

4.4 Media and Interaction Synchronization

An XML parser analyzes Input StoryML documents to build a structural object representation of the synchronization specification. This object structure mirrors the StoryML document structure to a schedule for the presentation. This schedule is managed by a timeline controller. The StoryML player implements a global timeline controller (Figure 8) for synchronizing media objects and interactions, which might be distributed over several interactors.

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 by starting several media objects on target interactors, as feed-forward information.

The dialog then requests the interactors to listen to the user events. Unlike MPEG-4 or SMIL, StoryML doesn't associate any user input to a specific media object, but to 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 change in the future, it will register this change to the timeline controller. In this way, the user interaction is synchronized.

5 REALIZATION

For demonstration purpose, a StoryML player was implemented on a PC and a robot, which respectively serve as an audiovisual interactor and a robotic interactor. The PC also provides services of a content portal, a StoryML parser, a timeline controller and an interactor manager. All these components of both sides are based on Java technology. Figure 9 shows the system architecture of the experimental implementation.

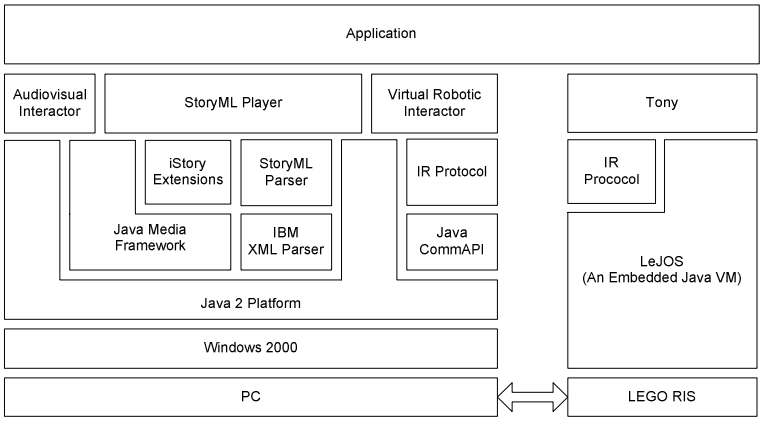

Figure 9. A StoryML experimental system architecture

5.1 PC Side

5.1.1 JMF

The JMF API [17] enables audio, video and other timed media to be added to Java applications and applets. JMF provides the StoryML player the means for controlling individual media streams in a way that is independent from delivery mechanisms, transport protocols, media types and encoding formats. The StoryML system uses JMF 2.1.1 mechanisms to synchronize media and deal with the timing issues.

5.1.2 StoryML parser

The StoryML parser is built on the base of the IBM XML Suite [18]. The StoryML parser traverses the entire tree of the DOM and constructs the internal representation of the StoryML objects.

Based on these objects, the StoryML player creates the Timeline Controller, the Interactor Manager and all the media objects. The Interactor Manager then prepares all the virtual interactors required by a StoryML document, and in turn the virtual interactors try to talk to the physical environment and negotiate with the physical counterparts to build the connection.

5.2 Robot side

5.2.1 LEGO RIS

The robot used for the demonstration is assembled using the LEGO Mindstorms Robotics Invention System (RIS) [19]. RIS includes two motors, two touch sensors, one light sensor, more than 700 LEGO bricks, and a programmable brick Robotic Command Explorer (RCX).

5.2.2 LeJOS

LeJOS [20] is indeed a small Java Virtual Machine that is downloaded to the RCX to replace the standard firmware from LEGO. A LeJOS program can use some standard Java libraries, such as java.lang, java.io, and java.util. It has libraries for control of motors, sensors, LCD and Buttons.

5.2.3 Tony

Figure 10 shows the robot, named Tony, used in the StoryML system. Tony is programmed in Java and simply can perform four different behaviors with its light, sound and movement output: it can be *at sleep*, *relaxed*, *in attention*, or *tired*. A request can be send from another device, for example, from a PC, which has an LEGO infrared tower connected to its serial port, to Tony's infrared receiver. Only when Tony is performing "in attention" and one of its touch sensors is being triggered, it will give immediate feedback with a beep and then send out an event to other devices, indicating which sensor has been touched.

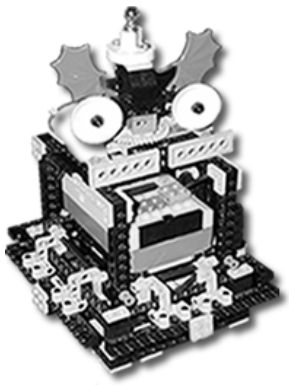

Figure 10. Tony

Tony is presented to a StoryML show in the StoryML system as an interface agent, and to the user as a companion. It watches the TV program together with the user and performs some requests from the program, i.e. renders some robotic behavior objects from the program. When the program gets Tony's attention, the user can press one of its touch sensors to react on the program. The user will get immediate feedback

from Tony and from the program as well. The feedback from the program is shown on one or many interactors.

5.3 Communication

A Virtual Robotic Interactor is created at the PC side as Tony's software counterpart. This Virtual Interactor keeps verifying whether Tony is present in the environment by sending handshaking messages. If Tony is not found, the Virtual Robotic Interactor will be visible on the screen with a standard graphic user interface. If Tony is present, the Virtual Robotic Interactor is hiding in the background. All the requests sent to the Virtual Robotic Interactor will then be transferred to Tony through the IR Protocol agent. The IR Protocol agent uses the Java Communications API to handle low-level details of IR communication with Tony's RCX through the IR tower.

6 CONCLUSIONS AND FUTURE DEVELOPMENTS

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:

- StoryML allows the separation of the content from the concrete physical devices
- A StoryML document specifies abstract media objects at a high level and leaves the complexity to the implementation of the rendering interactors.
- StoryML supports the automatic mapping of the same document to different environments, or a dynamic environment.

6.1 How StoryML Satisfied the Requirements

6.1.1 Distributed interfaces

StoryML has been defined as a solution for interactive story documentation, in which appropriate elements, i.e. <environment>, <interactor>, are dedicated to describing a desired environment which has multiple interactors involved. In the user's environment, the devices involved are self-contained and standalone. The StoryML player makes use of the PAC-based architecture, which emphasizes the independence of these devices and the communications between the system components.

The Abstraction component of a PAC agent is extended with MediaSource and MediaSink ports. A direct pipeline is built between a MediaSink and a MediaSource to improve the efficiency of the communication between these distributed agents while the control hierarchy of the system remains intact.

6.1.2 Context dependent interaction

The environment involves multiple devices with a variety of user interfaces. The configuration of such an environment is dynamic in both space and time dimensions. The StoryML player always first satisfies the desired environment described in the StoryML document with virtual software interactors, which is designed as an obligatory layer in its architecture. Software interactors function as the software counterparts for physical devices that are required by a StoryML document for interaction and presentation. The software interactors take the role of the physical device if it is not available. The user interaction can be switched between the hardware devices and their counter software interactors on the fly according to the current configuration of the system or the preference of the user.

6.1.3 Synchronized media and interaction

In StoryML, an implicit timeline is used for media and interaction synchronization specification. By comparing with the conceptual model of the interactive story, it is quite an intuitive way and the metaphor behind it can be easily understood. The StoryML player implements the timeline as an agent, i.e., the Timeline Controller, which monitors and manages the synchronization.

6.2 Strengths

Although the StoryML system is application or domain oriented - its focus is on a storytelling application for a limited target user group, the development approaches and results can serve as a framework for similar applications. As a good starting point, it paves a way towards generic solutions for serving interactive content in a distributed environment.

Many strengths of the StoryML system come with open technologies. The StoryML system is based on XML and Java technologies. Together, XML and Java technologies provide the StoryML system with strengths of simplicity, portability, and flexibility.

6.3 Weaknesses

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. Although some efforts have been made to improve the efficiency of the communication between the agents or interactors, this is a pitfall if the StoryML system is running on a poor network.

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 program which does not require any visual presentation, e.g. an interactive radio program.

6.4 Future Developments

For presenting interactive content in a distributed environment, a possible alternative approach is to distribute the entire StoryML document, instead of media objects, to all the interactors. Each interactor parses and distills the document and renders different parts. This approach does not make any assumption about the configuration of the environment. The synchronization of the media objects is also distributed, which will reduce the needs for centralized synchronization considerably. This will overcome weaknesses that the StoryML system has. It is very interesting, though it is not clear how the same documents rendered by different interactors can be kept updated simultaneously when the user interacts with one of them.

Acknowledgements

The author would like to thank Loe Feijs, Maddy Janse, Warner ten Kate, Emile Aarts and Don Bouwhuis for their insightful comments on earlier versions of the paper, and their supervisions on the work.

REFERENCES

1. Buford, J.F.K. Architectures and Issues for Distributed Multimedia Systems. *Multimedia Systems*. ACM Press, 1994, 45-46.
2. Duke, D.J.; Herman, I. A Standard for Multimedia Middleware. *Proceedings of the 6th ACM International Conference on Multimedia '98*. Bristol, UK. 381-390.

3. Eric S. Interactive Toy characters as Interfaces for Children. *Information Appliances and Beyond: Interactive design for consumer products.* Morgan Kaufmann Publishers, 2000.
4. NexTV. http://www.extra.research.philips.com /euprojects/nextv
5. Bukowska, M. *Winky Dink Half a Century Later.* Final report of the post-masters program: User-System Interaction. Aug. 2001. ISBN 90-444-0116-5. Or Nat.Lab TN 2001/353.
6. MPEG. Overview of the MPEG-4 Standard. Available at:http://www.cselt.it/mpeg/standards/mpeg-4/mpeg-4.htm.
7. W3C. SMIL 2.0 Specification. Available at: http://www.w3.org/TR/smil20/cover.html
8. 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*. 2000.
9. Mallart, R. Immersive Broadcast Reference Application. White Paper, Sept. 1999.
10. Herrmann, L. *Immersive Broadcast: Concept and Implementation.* Philips LEP Technical Report C 2000 748.
11. Bartneck, C., Affective Expressions of Machines, *CHI2001 Conference Proceedings*, Seattle, 2001.
12. Hu, J. *Distributed Interfaces for a Time-based Media Application*. Final report of the post-masters program: User-System Interaction. Aug. 2001. ISBN HuJun_ISBN90-444-0123-8
13. Hu. J. *Distributed Interfaces for a Time-based Media Application.* Nat.Lab. Report Rep 7204.
14. Buschmann, F., Meunier R., Rohnert, H. Sommerlad P., Stal M. Pattern-Oriented Software Architecture, A System of Patterns. John Wiley and Sons Ltd, Chichester, UK, 1996.
15. Calvary G., Coutaz J. Nigay L. *From single -user architectural design to PAC*: a generic software architecture model for CSCW.* Proceedings of the ACM CHI'97, Addison-Wesley, pp242-249.
16. Green J.S. Why is the Web so boring? Available at: http://www.news1st.org.uk/Interactive.htm.
17. Sun Microsystems, Inc. Java Media Framework API Guide. Available at: http://java.sun.com/products /java-media/jmf/2.1.1 /guide/.
18. IBM XML Suite of Java Beans. Available at http://www.alphaworks.ibm.com/alphabeans.
19. LEGO Mindstorms Robotics Invention System (RIS). Available at: http://mindstorms.lego.com/.
20. Solorzano, J. leJOS: Java based OS for Lego RCX. Available at: http://lejos.sourceforge.net/.