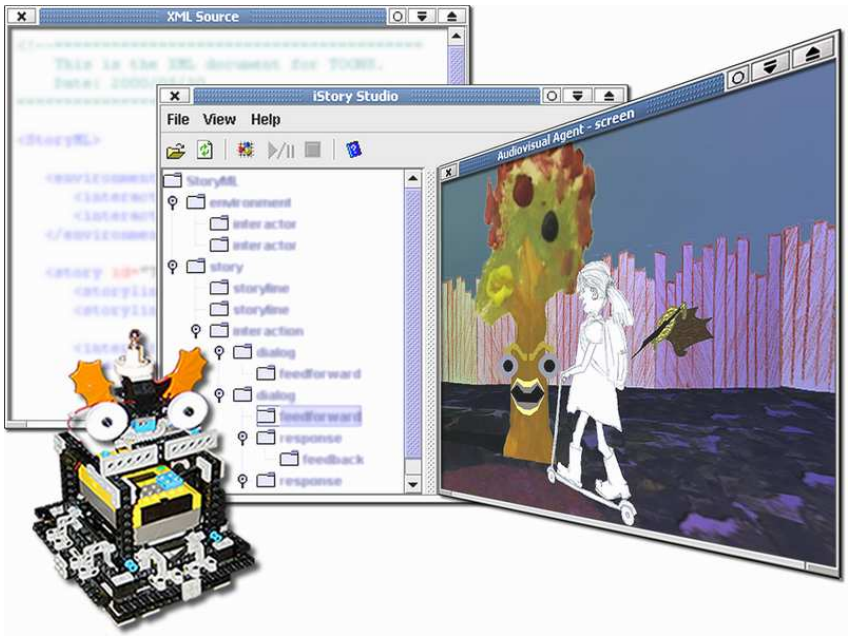
TU/e technische universiteit eindhoven

Distributed Interfaces for a Time-Based Media Application

Hu, Jun

ISBN 90-444-0123-8



/ stan ackermans institute,

DISTRIBUTED INTERFACES FOR A TIME-BASED MEDIA APPLICATION

by

Hu, Jun

A final report of the post-masters program
User-System Interaction

IPO, Center for User-System Interaction
Stan Ackermans Institute
Eindhoven University of Technology

2001

Supervised by

dr.ir. Maddy D. Janse (Philips Research)

ir. G.C. Sjir van Loo (Philips Research)

dr.ir. Raymond N.J. Veldhuis (IPO/TUE)

prof.dr. Yibin Hou (IPO/TUE)

ISBN-INFOMATION

"Distributed Interfaces for a Time-Based Media Application" / Hu, Jun;
Eindhoven: Stan Ackermans Institute. – Ill.
Final report of the post-masters program: User-System Interaction
With reference
ISBN 90-444-0123-8
Keywords: distributed interfaces / time-based media /

SUMMARY

DISTRIBUTED INTERFACES
FOR A TIME-BASED MEDIA APPLICATION

This project addresses the architectural issues of presenting interactive time-based media to distributed and networked devices. An experimental system was developed in which the interactive content was presented on a PC as well as on a robotic toy to create a more intuitive and pleasant interaction for children. This system, named iStory, was implemented using open technologies such as Extensible Markup Language (XML), Java Media Framework (JMF) and some other Java-based technologies. The key objectives are to investigate the feasibility of some new technologies, i.e., MPEG-4 and SMIL, and to develop a generic architecture for time-based media applications that are characterized by distributed interfaces.

This report first presents an overview of some time-based media technologies, with emphasis on the aspects of the content interaction. After introducing the requirements collected from a storytelling application for children, i.e., the TOONS application, a distributed content interaction model is developed to provide the context for the models used in the iStory system. Based on this model, this report analyzes the pros and cons of the current media technologies and some existing interactive TV interface architectures.

The iStory system is then presented. It implements a new architecture for storytelling applications. It provides functionalities for parsing, presenting and synchronizing the distributed media presentation and interaction specified by the StoryML, an XML-based language. The PAC-based interface architecture enables distributed interfaces and is extended for the time-based media use. The system develops MediaClock based and event based mechanisms for the synchronization. The system architecture of the experimental implementation is presented and major design issues are described.

In conclusion the relevance and applicability of the system is assessed and possible future developments are suggested. The findings indicate that while the iStory system can serve as a framework that can be refined and extended to fit the requirements of interactive time-based media presentation to distributed interfaces, there are still some issues that make their use for production purpose problematic.

Keywords distributed interfaces, time-based media

ACKNOWLEDGMENTS

I would like to express my gratitude to my supervisors, Dr. Maddy Janse, Dr. Raymond Veldhuis, Mr. Sjir van Loo and Prof. Dr. Yibin Hou. Their expertise, understanding and patience added considerably to my postgraduate experience. I appreciate their vast knowledge and skills in many areas. I saw Dr. Veldhuis used a blue sock to protect his palm-top computer, and "to shock snobs and moderate modern technologies." – He has different yet effective ways to manage problems, not only for his palm-top computer, but also in his expertise of streaming media technologies. Mr. Sjir van Loo is the system architect who told me that a car is more than a car; it is also a real-time operating system. His experience and insight saved me plenty of time from struggling with architectural issues of real-time systems. It was Prof. Hou who opened a window for me to the mystery of User-System Interaction. He always stands at high altitudes of the mountain called Information Technology so that his global view often leads me out of the jungle of technological chaos.

My special appreciation goes to Dr. Janse. She is the leader of the NexTV project. She created and managed a very nice project environment in which I had got enough freedom to concentrate on the research and design in my interests. She was so kind as to read the manuscript thoroughly and correct my English, even one day before I send it to the printer.

My collaboration with the NexTV group at Philips Research (Maddy Janse, Magdalena Bukowska, Marcelle Stienstra, Erik Bastiaans) has been very beneficial to me and I hope to them also. Magdalena has done a great job to collect the user requirements, which gives a piece of land for my hype of distributed interfaces to stand on.

The LEGO robot used in this project was originally assembled by Christoph Bartneck, a PhD student from the IPO, who is able to make a lovely and fantastic robotic toy out of a mess of the LEGO bricks within 5 minutes. Thanks to his patient explanations and hands-on helps, I got to know the LEGO robotics system and soon became a master.

I love the group of the USI 1999. There is a strong spirit of camaraderie in the group. I have enjoyed the chance to work within such a unique team that has shaped me as a designer and a person. We have contributed to each other's education and made our time at the IPO very pleasant. Thank you all for our philosophical and

political debates, exchanges of knowledge and skills, and international cooking parties during this two-year program: Michiel Alders, Taras Bahnyuk, Oleksii Bidiuk, Magdalena Bukowska, Hangjun Cho, Johannes Fahrenfort, Carsten Friedrich, Ingrid Halters, Karin van der Hiele, Robbert Kramer, Aliaksei Kuliashou, Sergey Lashin, Sergey Martchenko, Anita Morskate, Luuk Pernot, Weining Qi, Jeroen van Rooij, and Hongyu Wang.

The final acknowledgement goes to my family, who has provided me with a level of emotional support that can only be given, not expected.

CONTENTS

LIST OF FIGURES

ABBREVIATIONS

API	Application Programming Interface
AWT	Abstract Window Toolkit
BIFS	Binary Format for Scenes
CSS	Cascading Style Sheet
DMIF	Delivery Multimedia Integration Framework
DOM	Document Object Model
DTD	Data Type Definition
DVB	Digital Video Broadcasting
EPG	Electronic Program Guide
GUI	Graphic User Interface
HCI	Human-Computer Interaction
HTML	Hyper Text Markup Language
HTTP	Hypertext Transfer Protocol
IR	Infrared
IST	Information Society Technologies program of European Community
JFC	Java Foundation Classes
JMF	Java Media Framework
LDU	Logical Data Unit
MHP	Multimedia Home Platform
MIDI	Musical Instrument Digital Interface
MPEG	Moving Picture Experts Group
MVC	Model-View-Controller
NexTV	New media consumption in EXtended interactive TeleVision environment
NOP	No-Operation instruction
PAC	Presentation-Abstraction-Control
QoS	Quality of Service
RCX	Robotic Command Explorer
RIS	Robotics Invention System

RTP	Real-time Transport Protocol
SGML	Standard Generalized Markup Language
SMIL	Synchronized Multimedia Integration Language
StoryML	interactive Story Markup Language
UML	Unified Modeling Language
URL	Uniform Resource Locator
USI	User-System Interaction
VOD	Video on Demand
VRML	Virtual Reality Modeling Language
W3C	World Wide Web Consortium
XML	eXtensible Markup Language

1 INTRODUCTION

One day in the year 2007. 6:00pm in the afternoon. Xiaoxiao, a 12-year old boy, is watching a storytelling program TOONS showing on a wall in the living room, together with his little robot Tony. The lights in the room are changing the brightness following the story. Now in the story, a little girl enters a dark room. The living room becomes dark too. Xiaoxiao can't see clearly what's happening in that room and he doesn't like the darkness, so he adjusts the light besides him. Both the living room and the room in the story now are illuminated. In the story, the girl is wandering in front of two doors.

"Xiaoxiao, Should I help that lonely girl?" Sleepy Tony is woken up by the lights and seems attracted.

"Yes, go ahead." Tony approaches the wall and disappears from the living room. Suddenly he appears in the story. "Hi, Can I help you?" Tony asks the girl. "Yes. I can't decide which door to open."

"Left one, Tony!" Xiaoxiao doesn't know either, but the left door looks nicer. "Xiaoxiao wants us to go left." Tony opens the left door for the girl.

Behind the door there is a beautiful garden with colorful trees and puffy bushes. The sunset beams through the leaves and drops motley shadow into the garden and the living room as well. Xiaoxiao is surrounded by nice background music. He can hear birds singing their happiness around him.

"What a nice garden!" Xiaoxiao is now immersed in the story.

.....

1.1 BACKGROUND AND CONTEXT

People who have video viewing experience do have some "interaction" experience, but not more than simple operations to stop, start, pause, and fast forward/backward a video that is in progress. Some new technologies, such as MPEG-4 [22][29], SMIL [48][49], enable delivering interactive multimedia content to the consumers' homes. Possibly the greatest of the advance made by these time-based technologies is that people need no longer be passive. Instead, it allows end users to interact with media objects within a scene, whether they are natural or synthetic. These time-based media programs involve people in "doing and playing"

as well as "viewing". They can engage people in immersive experiences rather than just watching audio-visual programs [21].

The networked home will render these experiences. Philips Research has introduced a new notion of Ambient Intelligence that refers to electronic environments that are sensitive and responsive to the presence of people [23]. In this vision, the networked home will consist of clusters of embedded devices with user interfaces that extend the natural interactions. Distributed and networked interfaces will build up an immersive environment in the people's homes. By using physical interface agents, a more natural environment in which real-life stimuli for all the human senses are used, will give people more feeling of engagement [12]. Multimedia objects in the content can be audio, video, text and 2D/3D graphic objects. These objects can be even more abstract, such as, emotions, expressions and actions of characters. By making the best use of the home environment, the media objects can be distributed to several interface devices. For example, screens show the major part of the content with audio-visual objects, 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 program. Thus, the networked home will be the interactive media player.

During the past decade, research and development of time-based media technologies have increasingly focused on models for distributed multimedia applications [1][5][8][11]. The term “distributed multimedia” refers to the fact that the content sources of a time-based media presentation to the final user are distributed over a network. This report 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 QoS (Quality of Service) (cf. Figure 1-1).

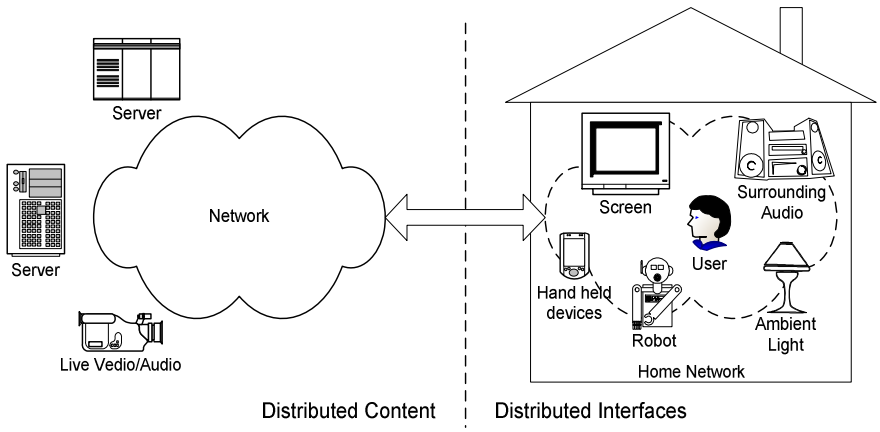


Figure 1-1.Distributed content and distributed interfaces

Distributed interfaces can raise several interesting research or design topics, e.g., distributed cognition [20], interface design, distributed content authoring tools and methods, etc. The focus of the work presented in this report is on interface architecture issues: **how to structure the system and content to support such distributed interfaces for time-based media applications**? The user interface controls the communication between user events at the distributed interface agents and the interactive objects of the content, and this transformation constitutes the user's immersive experience.

This work was done in the context of NexTV [19], a project funded by the Information Society Technologies programme of the European Community. It started in January 2000 and will last until December 2001. It involves 12 partners from all over the world: Philips Research (The Netherlands) as the project coordinator, Optibase (Israel), France Telecom R&D (France), T-Nova Deutsche Telekom (Germany), KPN Research (The Netherlands), TILIB (Italy), Sun Microsystems (USA), Sony Service Centre Europe N.V. (Belgium), Nederlands Omroepproductie Bedrijf (NOB, The Netherlands), GMD (Germany), ETRI (South Korea), and the Imperial Colledge of Science, Technology and Medicine (UK).

The major goal of the NexTV project is to explore the potential of new technologies, such as MPEG-4, DVB-MHP [13][26] , XML, for creating programs and environments in which users can actually interact with the content of the programs and be instrumental in creating their personal immersive experience [19].

The carrier for this 10-month USI final project is the development of an interactive storytelling application (TOONS) for children (age 8-12) [31]. The application is to enable children to create their own broadcast programs and environment by providing the means for exploration, manipulation and creation of TV program objects. The author of this report worked on this application together with Magdalena Bukowska, a USI student from the IPO, and three other people from the Media Interaction Group at Philips Research: Maddy Janse, the project leader, Marcelle Stienstra, a PhD student working on tangible interface devices, and Erik Bastiaans, a software engineer.

1.2 OBJECTIVES AND METHODS

One of the key objectives of this final USI project is to investigate the feasibility of some new technologies, i.e., MPEG-4 and SMIL in the context of a time-based media application that intends to take the advantage of distributed interfaces.

Based on the feasibility study and the user requirements, another objective is to develop generic interface architecture for these applications, in which the aspects of time-based and distributed presentation and interaction should be taken into account.

The NexTV project uses a user-centered design methodology for the fine-tuning of the requirements [31]. Finding the user and application requirements for the applications in the NexTV project is viewed as a process, which means that the application development was started from a set of general requirements that describe the overall scope of the application and that provide a starting point. These requirements will be fine-tuned during the project by means of feedback from users. In other words, the application development is considered as a process for acquiring and fine-tuning the requirements for users, applications, and system platforms.

The author joined the NexTV project in November 2000, when the overall requirements had been collected and documented. During the project, these requirements were refined by gathering input from the target user group, which was done by Magdalena Bukowska, while the author was doing literature research and feasibility study. Based on these refined requirements, a conceptual design of the interactive story was developed as an input for the system design, and current technologies and existing architectures were also evaluated. This phase of work established a theoretical and technical context for the system to be developed. Concepts, models and terminology were introduced to serve as reference points in the design and implementation.

The design and implementation of the system followed the Object-oriented methodology. The system was implemented using a specific set of emerging technologies, namely XML [46], Java [40][43] and JMF [37]. The intention was to create open-standards-oriented system architecture. More than on implementing a production-oriented system, the focus was on accessing the feasibility of the task and identifying critical design and implementation issues.

1.3 SCOPE OF THIS REPORT

Chapter 2 provides an overview of general technology issues related to time-based media systems, i.e. MPEG-4 and SMIL. A content interaction model is introduced to generalize the concepts of these time-based media systems. Chapter 3 describes the conceptual model of the TOONS, an interactive storytelling application. The requirements from the users and the project are briefly introduced. Two existing digital TV interface architectures and the existing technologies are analyzed in the context of these requirements. The content interaction model is refined for the distributed interfaces.

Chapter 4 first introduces a generic interactive story model together with its object-oriented model. The XML documentation (StoryML) enables the iStory to be served, received, and processed over the network. Chapter 5 provides an overview of two different interface architectures, i.e. MVC and PAC, and an analysis of their pros and cons in the context of time-based media applications. The PAC is selected and extended for the purpose of this design. The architecture of the StoryML player (iStory) is given at the end of this chapter. The iStory system uses two synchronization mechanisms based on a timeline controller, i.e., MediaClock based and event based synchronization. Chapter 6 first introduces some concepts of multimedia synchronization, and then in the context of these concepts, describes the synchronization mechanisms used in the iStory system.

Chapter 7 gives the overview of the system architecture of the implementation with a PC and a LEGO robotic toy, and some technical details and difficulties as well.

In conclusion, Chapter 8 summarizes insights gained and suggests possible paths for further work.

2 OVERVIEW OF TIME-BASED MEDIA SYSTEMS

2.1 TIME-BASED MEDIA [1]

Any data that changes meaningfully with respect to time can be characterized as time-based media. Audio clips, MIDI sequences, movie clips, and animations are common forms of time-based media. Such media data can be obtained from a variety of sources, such as local or network files, cameras, microphones, and live broadcasts.

A key characteristic of time-based media is that it requires timely delivery and processing. Once the flow of media data begins, there are strict timing deadlines that must be met, both in terms of receiving and presenting the data. For this reason, time-based media are often referred to as streaming media - they are delivered in a steady stream that must be received and processed within a particular timeframe to produce acceptable results

2.1.1 Content types

The format in which the media data is stored is referred to as its content type. QuickTime, MPEG, and WAV are all examples of content types. Content type is essentially synonymous with file type - content type is used because media data is often acquired from sources other than local files.

Among the principle characteristics of multimedia systems is the capability to process and present these content types, such as audio, video, image, text, 2D and 3D graphics, and animations, to facilitate the communication of a specific form of information. Instances of specific content type are often referred to as media objects.

A high-level distinction is made between time-dependent and time-independent content types [3][50]. Time-dependent media objects are handled as streams of logical data units (LDUs) [3] or samples, which are presented according to the implied or specified time dependencies of the object [25]. In the common case of continuous media, successive data units are presented at constant intervals. Typical examples of continuous media are audio and video streams, which are rendered by playing back audio samples or video frames at a specific rate.

[1] Some of the introduction about time-based media is edited from [37].

Time-independent media, typically text or images, are not intrinsically related to any points in time. The playback of such media in a multimedia presentation is controlled by externally imposed time dependencies. For example, an image may be displayed for the duration of a particular audio stream, which means that the temporal endpoints of the audio stream are imposed on the image. It is normally possible to pre-fetch time-independent media objects completely before they are required, which makes the related presentation issues trivial in comparison with time-dependent media.

2.1.2 Media streams

A media stream is the media data obtained from a local file, acquired over the network, or captured from a camera or microphone. Media streams often contain multiple channels of data called tracks. For example, a Quicktime file might contain both an audio track and a video track. Media streams that contain multiple tracks are often referred to as multiplexed or complex media streams. Demultiplexing is the process of extracting individual tracks from a complex media stream. A track's type identifies the kind of data it contains, such as audio or video. The format of a track defines how the data for the track is structured.

A media stream can be identified by its location and the protocol used to access it. For example, a URL might be used to describe the location of a QuickTime file on a local or remote system. If the file is local, it can be accessed through the FILE protocol. On the other hand, if it's on a web server, the file can be accessed through the HTTP protocol. A media locator provides a way to identify the location of a media stream when a URL can't be used.

2.1.3 Media delivery

Media streams can be categorized according to how the data is delivered:

Pull - data transfer is initiated and controlled from the client side. For example, Hypertext Transfer Protocol (HTTP) and FILE are pull protocols. This is also known as progressive streaming or progressive download, it refers to online media that users can watch as the files are downloaded. The user can see the part of the file that has been downloaded at a given time, but can’t jump ahead to portions that haven’t been transferred yet. Progressive streaming files don’t adjust during transmission to match the bandwidth of the user’s connection like a real-time streaming format

Push - the server initiates data transfer and controls the flow of data. For example, Real-time Transport Protocol (RTP) is a push protocol used for streaming media. Similarly, the SGI MediaBase protocol [35] is a push protocol used for VoD (Video on Demand). Real-time streaming refers to technologies that keep the bandwidth of the media signal matched to that of the viewer's connection so that the media is always seen in real-time. The word real-time differentiates this type of streaming from HTTP streaming delivery. Dedicated streaming media servers and streaming protocols are required to enable real-time streaming. Real-time streaming delivery always happens in real-time, so it is well suited to live events. It also supports random access of material, so the user can fast forward to other parts of the movie, which may be useful for presentations and lectures. In theory, real-time streaming movies should never pause once they start playing, but in reality, periodic pauses may occur.

2.2 MEDIA APPLICATION DOCUMENTATION

Multimedia application is generally organized and managed in documents. A document describes the components of a multimedia presentation together with information such as associations, temporal and spatial relationships. Documentation standards are essential for delivering applications across heterogeneous platforms. The concept of "multimedia standard" implies a declarative document format to represent an audiovisual scene as a composition of audio and visual objects with specific properties and behavior, in both dimensions of space and time.

MPEG-4 and SMIL are such a standard solution. They originate from different communities and address a somewhat different "multimedia profile". However, the bundle of concepts they share makes them suitable to convey interactive time-based media applications to users, such as EPG's, interactive stories, sports events and so on.

The general issues that need to be addressed by a comprehensive media documentation model can be summarized as follows:

- Synchronization
- Integration of multiple media types
- Composite media objects
- Media object addressing
- Hyperlinking
- Input model for interaction

2.2.1 MPEG-4

MPEG-4 provides new coding technologies for the manipulation, storage and communication of multimedia objects including video, speech, audio, texture, graphics, text and animation. The standard is not a direct improvement of MPEG-1 or MPEG-2 coding standards; rather it defines a different way of integrating and interacting with scene content. It provides a set of technologies to satisfy the needs of authors, service providers and end users [29].

In MPEG-4, the elements in a scene can be described by using a tree-like structure. At the root of the tree we have the entire (or Composite) scene. This composite scene is broken down into individual objects (such as a person or a movie clip) and then into atomic structures called Media Objects (such as the person's video and audio channels), which make up the leaves of the tree. MPEG-4 standardizes a number of primitive media objects. It is capable of representing both natural and synthetic content types, such as still images, video objects, audio objects, text, graphics, talking synthetic heads, synthetic sound etc.

MPEG-4 uses a language called Binary Format for Scenes (BIFS) to describe and dynamically change the content of the scene. It is through BIFS commands that objects in a scene can be added, removed, or have their visual and acoustic properties changed without changing the object itself. BIFS can be used to animate objects by sending a BIFS command and to define their behavior in response to user input at the decoder.

The event model of BIFS uses the VRML concept of ROUTEs to propagate events between scene elements. ROUTEs are connections that assign the value of one field to another field. ROUTEs combined with interpolators can cause animation in a scene. User inputs can be detected with sensors. Sensors generate events when the user interacts with an object or a group of objects. Sensors also react to changes in the environment and changes of time, and drive the time of animations.

MPEG-J is a set of APIs (Application Programming Interfaces) that allow Java code to communicate with an MPEG-4 player engine. By combining MPEG-4 media with safe executable code, content creators may embed complex control and data processing mechanisms with their media data to intelligently manage the operation of the audio-visual session.

2.2.2 SMIL

SMIL is a proposed recommendation developed by the W3C Synchronized Multimedia (SYMM) Working Group. It aims to bring synchronized multimedia to the Web without requiring expensive authoring tools. All the descriptors in the SMIL specification follow an HTML-like format. These syntax elements provide an extension on top of HTML and XML that tells the browser how to position in time and space, and how to link to other content. The syntax of SMIL is defined using the DTD (Document Type Definition) notation. The formalism is human readable.

Using SMIL, authors can create a multimedia presentation in terms of layout, temporal behavior, and events associated to timers and anchor selections. A SMIL document essentially includes a "layout" and a "body".

The layout specifies a set of "regions" on the screen. The region element controls the position, size and scaling of media object elements. The author can select between SMIL "basic layout" and expressing layout in CSS2 (Cascading Style Sheet).

The body contains information related to the temporal and linking behavior of the document. A document structure relies on a tree-like aggregation of "synchronization elements", which type can be sequential or parallel. These elements synchronize the presentation of their children. At the leaf nodes there are "Media object elements", which manage the presentation of animation, audio, image, video, text and text-stream elements. Elements activation can be deferred to the occurrence of an event (e.g. generated by a video clip). Links are actuated only by the user and support navigation between objects. A link can activate a new document or a single object within a document.

SMIL represents the Web perspective, which demands capability to synchronize streams with other elements in the framework of a single application. A native support by Web browsers will allow widespread availability of applications, and, thanks to the text format, a search engine is able to find a SMIL document on the Web. The simple syntax allows building applications without the need of a sophisticated authoring tool.

2.3 CONTENT-BASED INTERACTION

2.3.1 Three levels of interactivity

In general, there are three levels of interactivity in the context of interactive content: Content retrieval, content navigation and content-based interaction (cf. Figure 2-1).

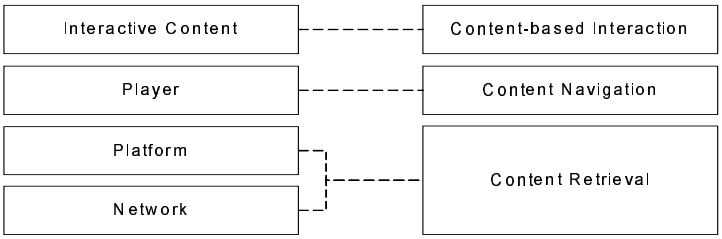


Figure 2-1.Three levels of interactivity

At the level of content retrieving, the user can retrieve the content by sending a query to the system or the system can filter desired content out with user profiles. The user can select one from many results that the system provides. Selecting a TV channel or searching a website are good examples of content retrieving.

The user can also make use of the means provided by the content player itself to navigate the content, e.g., pause, fast forward/backward a video, or go back/forward to, scroll up/down a web page.

Interactive content contains not only rich multimedia information (media objects), but also application logic that can be declarative and/or procedural. The later makes the output of interactive content affected either by the user directly, or by responding in some way to its "knowledge" of the user or users (i.e. of their behaviors and characteristics). In this way, the content is interactive:

Interactive content = data (media objects) + control (application logic)

This report concentrates more on the content-based interaction, which means users can manipulate and influence the content they receive, through the interface authorized inside the content, presented by a player.

2.3.2 A content interaction model

In [9], a HCI (Human Computer Interaction) model is presented that introduces the taxonomy of the VRML application interactivity. Inspired by this model, a content interaction model is developed below to generalize the content-based interaction.

In a co-operative user-system process, both the user and the system possess information that needs to be exchanged. The user wants to either import his knowledge to the system or tell the system what data he wants to retrieve from it. The system is supposed to deliver the requested information to the user. It can send status messages and ask the user for more information.

In principle, all system input and output requires the following two-step transformation: abstraction and interpretation for input, representation and presentation for output, which in turn forms user-system dialogs [9]. The content player and I/O devices form the bridge between the physical media and the digital content carried inside the system.

The physical signals received through the input devices are highly fragmented and contain no explicit semantics. The input device detects physical signals and describes them in a digital form. An abstraction function is necessary to recognize the larger symbolic forms to give presentation of the inputs as user events. The result of the abstraction function is still merely a representation and not the concept behind the representation. To retrieve the concept, the system must use an interpretation function. The retrieved concepts or commands can then be further processed or stored in some internal system representation [9].

An opposite two-step transformation process is necessary at the output side [9]. For example, when the system has to convey certain content to the user, the content must be represented in a certain form. This could be a representative data structure in an abstract form. The application logic in the representation is parsed and will control the system behavior and processes, which makes the content interactive. The media content in the representation subsequently be specified or rendered into a raw description of the presentation that the output devices can handle.

At the user's side, Norman's seven stages model gives a clear overview of interaction [32]. The user might perform similar presentation-representation manipulation of information, which is not shown in detail in Figure 2-2:

- Form a goal – the environmental state that is to be achieved;
- Translate the goal into an intention to do some specific action that ought to achieve the goal;
- Translate the intention into a more detailed set of commands – a plan for manipulating the interface;
- Execute the plan – present the inputs to the system by the input devices;
- Perceive the state of the system – perceive the output presentation of the system by the output devices;
- Interpret the perception in light of expectations;
- Evaluate or compare the results or the intentions and goal.

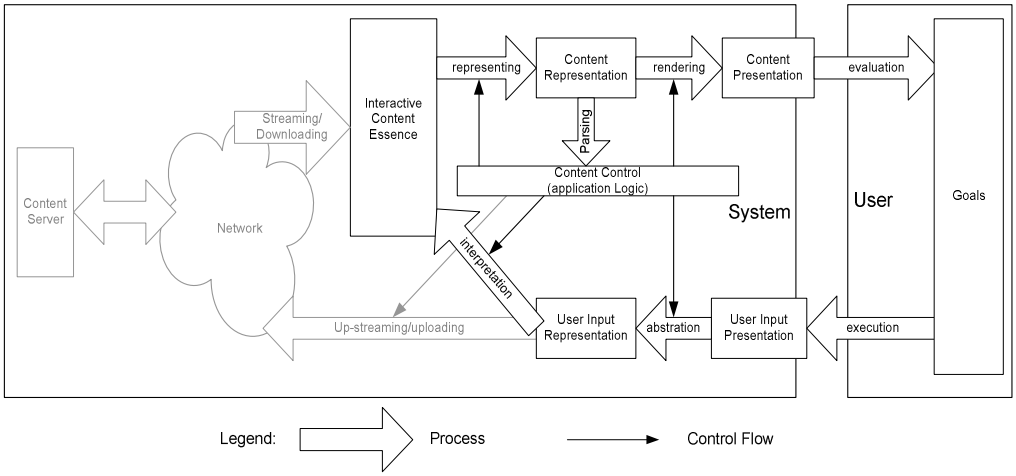


Figure 2-2.Content interaction model

Figure 2-2 shows the concrete process information transformation that the system performs during a user-system dialogue. Either the user or the system can initiate the dialogue circle. For the user's side, Figure 2-2 simplifies Norman's seven stages to a simple form: The user evaluates the information conveyed by the content presentation and reacts with user input presentation to fulfill certain tasks or goals.

Interactive content can be stored locally or come from a network, or from a combination of both. User input information might be sent to the content server over the network to influence the content according to the application semantics. In most of the cases, network transportation of the content and the user input is transparent to the user and the application, unless slow network transportation delays the response of user input or the application wants the user to be aware of the difference between local and remote access of content.

2.3.3 An example: MPEG-4 interactivity

The most significant aspect of MPEG-4 is that users no longer need to be passive in the consumption of Multimedia. The extent of interactivity with most common multimedia applications is the ability to stop, rewind, fast-forward, play and pause. In MPEG-4, the user can directly interact with objects within the scene. The implementation of the media objects as separate entities, combined with separate scene descriptors, allows a user to potentially move around in the scene, move objects around, modify attributes of each object such as color, or add and remove objects from the scene. The possibilities of interaction are nearly endless, with the limits resting solely on the multimedia author. This is known as client-side interaction [29].

Interaction behavior is specified in the scene description information encoded in the BIFS stream, which implements declarative application logic. This information specifies the modification of attributes of scene objects according to certain user actions. User actions on media objects (such as mouse clicks, etc.) are passed to the data controller.

The data controller either traverses the scene graph to derive the nodes of these media objects for firing the appropriate event handlers to process these events, or sends the user events back to server for client-server interaction.

Figure 2-3 shows the interactive architecture of the MPEG-4 client.

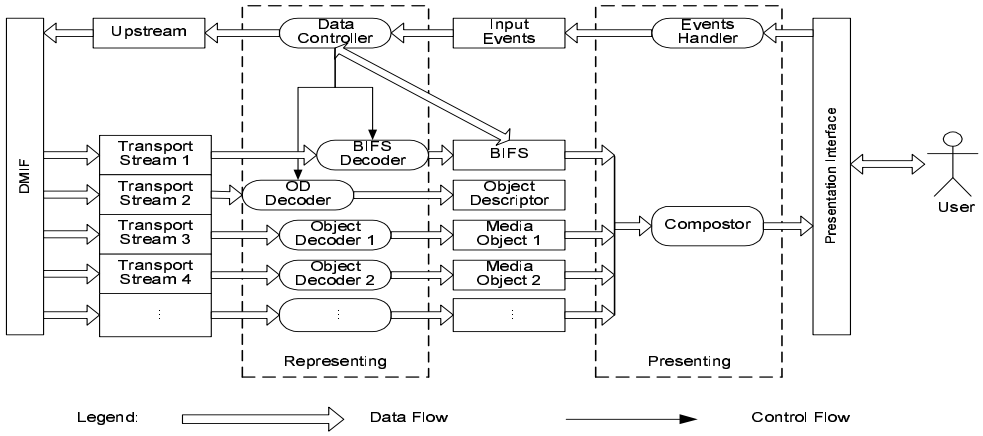


Figure 2-3.Interactive architecture of MPEG-4 client

In MPEG-4 version 2, MPEG-J introduces procedural functionality for creating interactive applications. It can also be streamed to and decoded at the client side. It will be passed to the data controller and executed to perform higher interactivity.

MPEG-4 interactivity provides standardized bricks to build up interactive applications. Depending on how the content and user input are organized (application logic) and how the content and user input are presented (I/O devices), different levels of content interactivity can be achieved in MPEG-4 applications.

3 WHEN TECHNOLOGIES MEET REQUIREMENTS

3.1 USER REQUIREMENTS

The overall goal of TOONS is to develop an interactive storytelling application in the broadcast domain to enable young children in the age group 8-12 years old to [30][31],

- Create their own broadcast environment by positioning them the role of program maker.
- Provide them with tools to interact with the broadcast environment;
- Make personal content.

To achieve this goal, a user-centered methodology is used in the NexTV project. The story and interaction scenario are created in cooperation with the end-users. According to the goal of the TOONS application, Magdalena Bukowska conducted user trials to collect the input from the target user group. Some raw video materials were edited from an "Old House" scenario created by NOB, one of the NexTV project partners, the Dutch broadcast production company. She created a Macromedia Director® movie out of the assets provided by NOB and additional 2D hand drawings, with limited interaction. The input of the users was used to create the interactive story and to develop a conceptual model of the story. (More details about this part of work can be found in the USI final report of Magdalena Bukowska [6])

3.1.1 Conceptual model of TOONS

3.1.1.1 Two types of interaction

Depending on how the interaction is initialized, two different types of interactions can be involved in the TOONS application (cf. Figure 3-1):

1. Initialized by the application.

The users can only give their responses to the program at a certain moment or in a certain period. In this case, the application initializes the interaction by starting a feed-forward and feedback loop. The application initializes the dialogue by providing feed-forward information such that the user is invited or enticed to give responses. The user evaluates this feed-forward information and compares this with his/her goal to make up their mind about their response. If the user responds, immediate feedback is expected to confirm the

user's input. The interaction will result in immediate or delayed changing of the story content.

The dialogue has to be finished within a certain period. If the user does not give any response in this time period, the application will proceed with a default response.

2. Initialized by the user.

The user can interact with the application as long as the program is playing, although the available functions may vary throughout the program. In this case, the user initializes the dialogue by activating the interface. The following dialog procedures are similar to the dialogue procedures in the case that the interaction is initialized by the application.

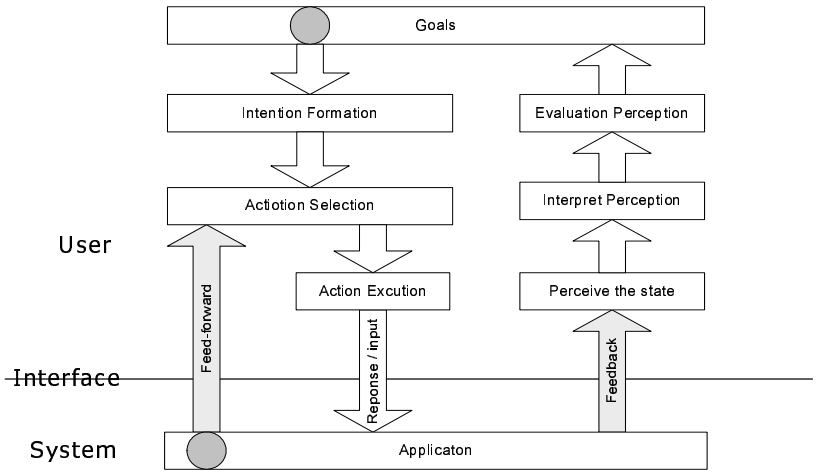


Figure 3-1.Two types of interaction[2]

The TOONS application is a broadcast program which tries to push content to the user according to the user's interest. In this sense, most of the interactions are initialized by the application and the dialogs between the system and the user appear linear.

Note that the term "feedback" here refers to the immediate information that is given by the system to confirm the input of the user, whereas the term "results" refers to the immediate or later changes of the story content affected

[2] This figure is based on Don Norman's seven stages model [32]. Two gray dots indicate possible starting points of a dialog.

by the user input. This implies that, when the interaction causes immediate changes in the content, the results may be used as feedback.

3.1.1.2 Interaction in TOONS

Figure 3-2 shows the conceptual model of TOONS. 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 decide. For example, to open different doors, the user can knock on different buttons on a console. Different decisions at the decision points will lead to several different storylines depending on the choices the user made. Within the storylines, the user can also customize the unfolding story by adding content, for example, a companion or a drawing they made. 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.

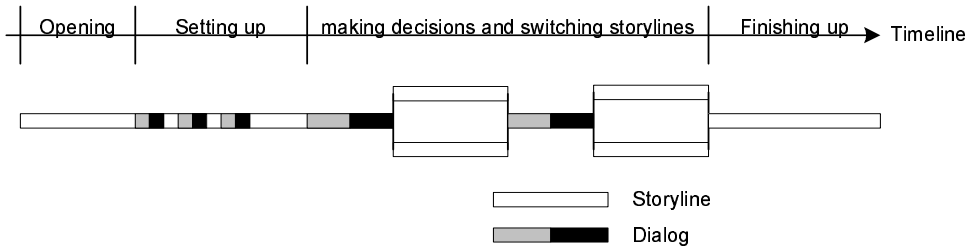


Figure 3-2.Conceptual model of TOONS

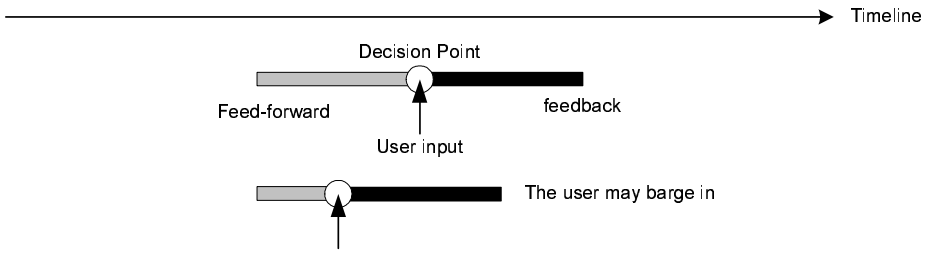


Figure 3-3.Simplified view of a dialog

A simplified view of a dialog is shown in Figure 3-3. Notice that the user may barge in to respond to the program at any point as long as the feed-forward information is playing.

By giving a certain amount of feed-forward information, the TOONS application will entice the user to,

- Make decisions by selecting objects to switch the storylines immediately or at a later point in time.
- Customize the story by adding or deleting objects or by changing their properties. This will cause an immediate change in the scene which might influence the storyline.

The responses from the user might cause,

- Immediate feedback to confirm user input. For example, a voice saying "Yes, you made it! The left door will be opened for you." after the user made the choice to open one of several doors.
- Immediate changes on the objects in the scene. For example, the user can choose their favorite color for the figures' clothes and once the selection is made, the color will be changed as both result and feedback.
- Later changes on the objects in the scene. In TOONS, the user might like the main character to have a companion. Once the user decides so, the companion will appear in the scene.
- Immediate switch among several available storylines. It can be treated as immediate feedback to user input as well.
- Later switch among several storylines. According to the decisions or choices the user made earlier, the application will tell different stories, starting at specific moments later in the story.

3.1.2 Tangible interface devices

"The children have tangible interaction tools in their possession to interact with the on-screen objects and streams. These interaction devices will provide feed-forward and feedback in order to facilitate the interaction with the story. Feed-forward can be obtained for example through light or sound in the interface device to indicate what can be activated, what actions can be undertaken, and how the actions can be achieved. This information must be broadcast by the broadcaster along with the story: what actions can be undertaken at what point in the story. To enhance its effect, the story should also supply information needed for the interaction, be it clues, voice-overs that tell the users that a decision point is at hand or similar things.

The feed back provided by the interface device should be given immediately after actions are undertaken by the children. This can be in the interface

device itself, through sounds (*'click'* when a button is pressed) or light (an illuminated button that has been activated). In any case, information about the result of the user action should always be immediately presented, so that the users know that their actions did have some effect." [30]

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 [6], Karolina, a 12-year old girl, suggested a robot as the tangible interface device (cf. Figure 3-4). "*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.

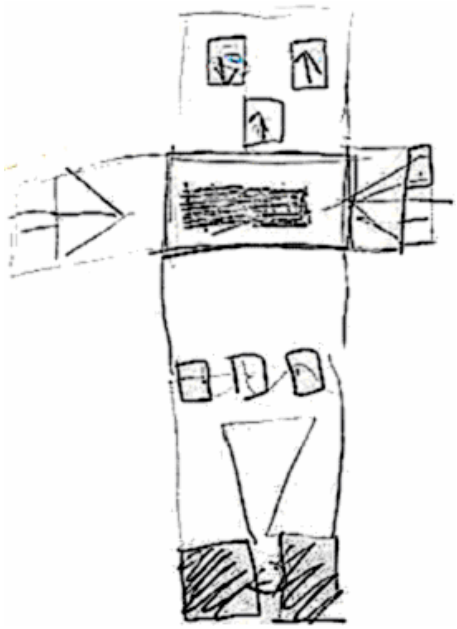


Figure 3-4.Karolina's robot

3.1.3 Four types of decision points [3]

The TOONS application focuses on four different types of decision points. These decision points are derived from enabling features of the MPEG-4 object representation. These decision points are:

[3] This part of content is edited from Marcelle Stienstra's descriptions and drawings, appears in the NexTV deliverable "Application Version 1" [30].

1. Influencing the story line by choosing an object. The story line depends on the object that is chosen by the user, for example a key or a shovel. If the user, for example, chose the key, the character in the story would move into the secret room through the door with the large lock that the user were now able to open. It is assumed that this type of decision point will support children that are interested in action and adventure in the plot of a story.

2. Influencing the story line by choosing an emotion, i.e., changing a property of an object. The user can select an emotion for a specific character in the story. This emotion can be a happy or a sad mood. Depending on this mood a different story line will be followed. The moods can be attached to characters but also to objects, rooms and so on. It is assumed that this type of decision point will support children that are interested in the social and emotional developments of characters in a story.

3. Adding characters to the scene, i.e., adding an object. The user can add a character to the story. This character will appear in the story, but does not influence the story line. It is assumed that this type of decision point will support children's fantasy and the ability to create one's own story in the context that is provided by the broadcast story.

4. Influencing the storyline by forming a team, i.e., changing the relation between objects. The user can form a team of two characters from a number of characters. Depending on which characters are in a team, a different story line will be shown. It is assumed that this type of decision point will support children that are interested in the social and emotional development of characters in a story.

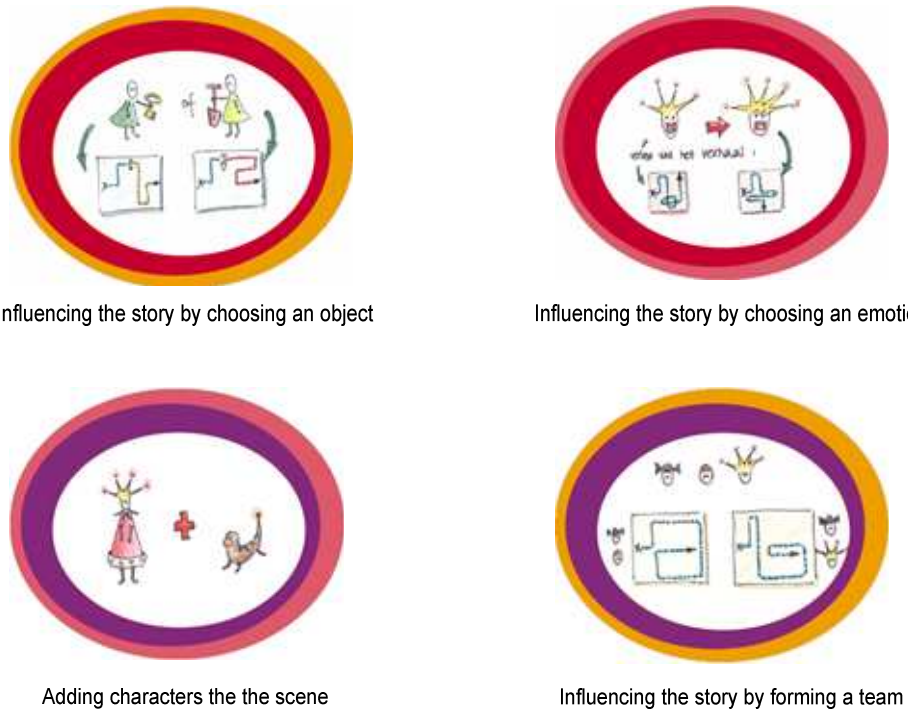


Figure 3-5.Four types of decision points

3.2 REQUIREMENTS FOR THE APPLICATION ARCHITECTURE

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

3.2.1 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. These networked devices will provide the users with more immersive experiences.

3.2.2 Context dependent interaction

Here the term "context" means the environment availability, application context, and user preference.

The target system platform can vary from a simple TV set with a set-top box, to a complicated home network environment. The interface devices can be as simple as a remote control or as complex as a robot with multi-modality interfaces. Furthermore, 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 light instead of pressing up or down buttons on a remote control (see the story introduced in chapter 1).

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.

For the demonstration purpose, the TOONS application implements only some of the above. For future extensions, however, all of the above has to be taken into account to arrive at generic system architecture for similar applications.

3.2.3 Synchronized media and interaction

In an interactive media application, not only the media, but also the interactions are time-based 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 or 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.

3.3 DISTRIBUTED CONTENT INTERACTION MODEL

Distributed Interfaces introduce new challenges to the content interaction model presented in section 2.3 (cf. Figure 2-2). This model implicitly assumes that only one device, e.g., a PC or a television, is used for the presentation. Although multiple input devices may be included in this model, such as a keyboard, a mouse and a remote control, these devices are treated as accessories for user input and fully controlled by the presentation device.

Figure 3-6 shows a new content interaction model in which multiple interactors have been considered. An interactor is a software or hardware entity with independent and different capability for data processing, and with different input and output components which form a user interface. These interactors are networked together and cooperate with each other to present the interactive content.

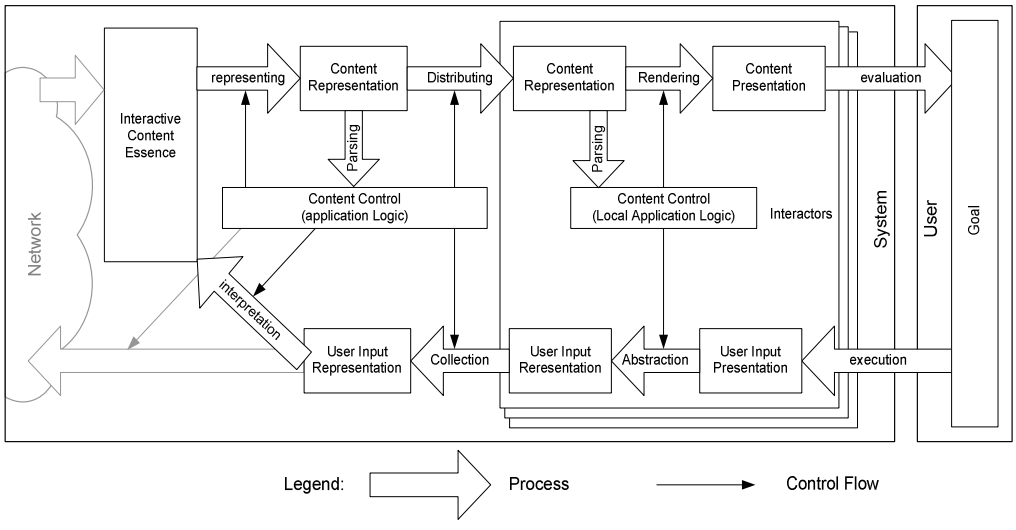


Figure 3-6.Content interaction model with distributed interfaces

The application logic is parsed from the content representation. According to the application logic the content is distributed to different interactors and the user events (user input representation) are collected from these interactors. Under the control of the Local Application Logic and parsed from the local content, each interactor performs the two-step transformation process which was described in section 2.3: abstraction and interpretation for input, representation and presentation for output. The user interacts with these interactors and in turn with the interactive content.

This approach challenges current 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. BIFS based MPEG-4 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. The SMIL 2.0 introduces *MultiWindowLayout* module, which contains elements and attributes providing for creation and control of multiple top level windows [49]. This is very promising and comes close to the requirements for 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 a software interactor with the same capability.

The TOONS application intends to make use of multiple interactors 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.

3.4 SOME EXISTING INTERFACE ARCHITECTURES

3.4.1 Future TV

Future TV is a research project which aims to study the digital television as platform for new interactive multimedia services [14]. One of the goals in this project is to implement the user interface of digital television by using well-known APIs defined for the DVB Java platform. A paper has been published describing how to develop a Java user interface, which includes not only graphics but also time-based media [34].

Figure 3-7 illustrates this basic user interface structure for interactive television services or applications on a set-top box. The Java 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. Graphics

in this case means the visual presentation of interface widgets. Broadcasting content consists of video, audio, subtitles, teletext, and data. This structure serves a clear architecture for users to manipulate content-related information, such as EPG, Home Shopping, TV chat and so on.

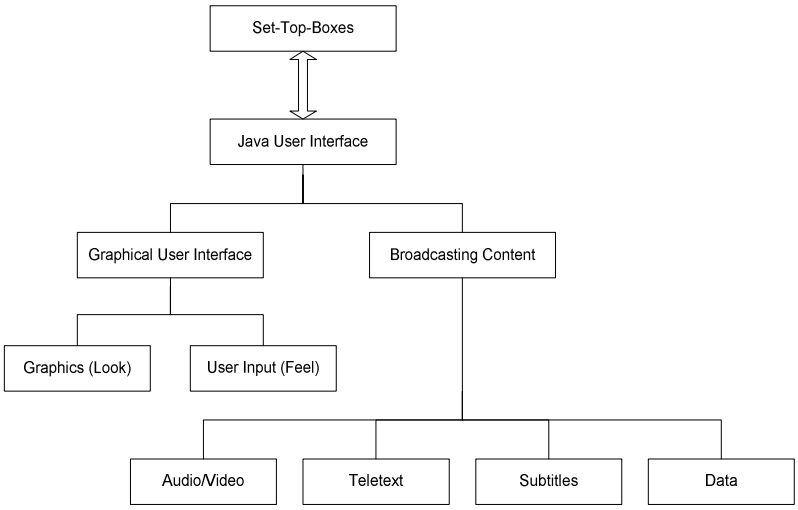


Figure 3-7.Basic structure of the user interface in digital TV

In this structure, however, the GUI is not a part of the content. The user may select different content by manipulating GUI widgets, but the interaction does not influence anything inside the content. It facilitates content retrieval and navigation, rather than content-based interaction. 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.

This structure has no concern for networked multiple devices and distributed interfaces, nor synchronization between the media and the interaction.

3.4.2 Immersive broadcast

Immersive broadcast is a generic term for interactive multimedia applications mainly targeting enhanced digital television programs [27]. An "immersive broadcast" application for sports events is presented in [16]. 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.

The application is composed of the following components [16][27]:

- Views. Views are live components that occupy the full screen, often consisting of a full screen video, associated audio channel, and some additional assets such as text, graphics, and images.
- Highlights. Highlights are stored views that are assessed on demand and provide instant-replay of important events. They are similar to views but do not contain live content.
- Standing & Results Overlay. It is a pop-up screen overlaid on any other screen to provide a snapshot of the current standings and results. The content of this screen is dynamic and the transparency effect is required.
- User interface for navigation & browsing between or through views, highlights, and overlays
- Alerts. Alerts are user interface elements that pop up unexpectedly to give direct access to a view or a highlight when a noteworthy event takes place.

These components can be categorized into content presentation components (views, highlights and the overlay) and content navigation components (cf. Figure 3-8). Compared to the structure shown in Figure 3-7, 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.

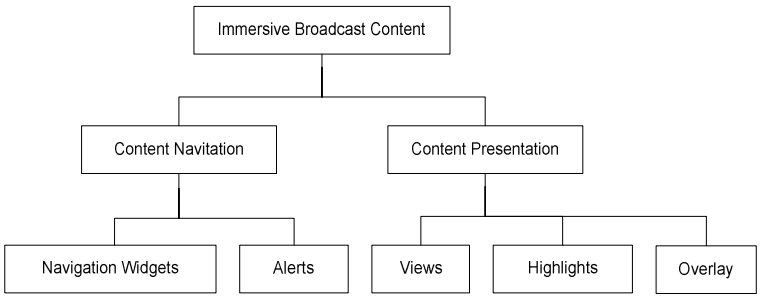


Figure 3-8.Components of an "immersive broadcast" program

This structure doesn't take into account the distributed presentation and interaction either; the user interaction still stays on a level of content navigation, which will not change what's in the content.

In short, none of the existing interface architectures can account for the distributed presentation of content and the actual interaction of users with the content itself. For this we need a different approach. This approach will be presented in the next chapters.

4 STORYML

To achieve the goal of TOONS and satisfy the requirements described in chapter 3, the first question which has to be answered is how to describe such an interactive story that will be played in a distributed environment. A new documentation technology has to be developed, since existing open standards can hardly describe an environment which involves many distributed interfaces instead of only one multimedia terminal.

This chapter presents StoryML, an XML based specification language for interactive stories. It is aiming at a technical solution for the interactive story representation, which can be served, received, and processed over the network and finally played in a distributed environment.

4.1 OBJECT-ORIENDED STORY COMPONENTS

The development of StoryML is based on the conceptual model of interactive stories, presented in chapter 3. This model was developed in the coordination with both content producers and final users [6]. StoryML reflects directly many concepts from this model. One objective of doing so is to make StoryML an easy authoring language for content producers, with a higher level of abstraction which is independent of media representation technologies.

The StoryML elements are derived from the conceptual model of interactive stories and the requirements for distributed interfaces by using an object-oriented approach.

4.1.1 Generalized interactive story

Figure 4-1 shows the generalized interactive story model which is directly derived from the conceptual model (cf. Figure 3-2). An interactive story consists of many storylines and the user can influence these storylines. Many linear dialogs compose the interaction between the user and the story. A dialog always starts with the system conveying certain feed-forward information the user, the user makes decisions or choices and then the system shows immediate feed-back information to the user. The dialog results in switching between the storylines, or changes to one or many storylines. Compared with the conceptual model, this model clearly expresses the hierarchy structure of the interactive story. It no longer emphasizes different phases of a story, which is related to the implementation of a specific story.

This model explicitly defines a story line as a primitive story component which has the same temporal dimension and behavior with the story, that is, starting or stopping a story means starting or stopping these storylines at the same time.

The feed-forward and feedback components are separated from the storyline as primitive components because they have different temporal behavior. Different from the storylines, when to start a feedback component or when to stop a feedback component depends on when the user will respond during a dialog.

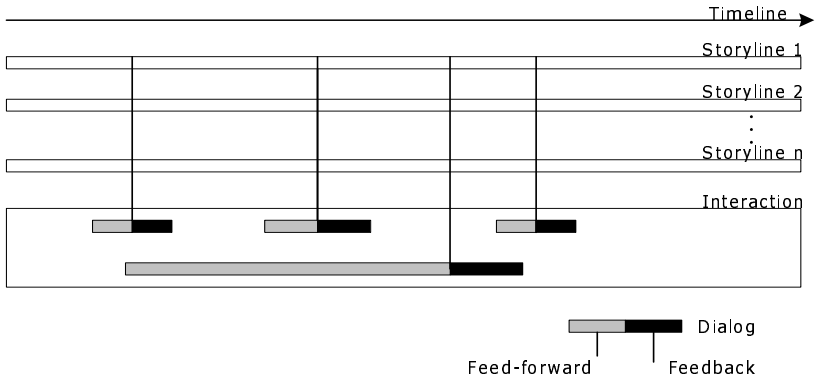


Figure 4-1.Generalized interactive story derived from the conceptual model

4.1.2 Environment

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 and so on. 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 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. According to its expertise, these interactors can be categorized into different types.

An environment is then defined as a dynamic configuration of many interactors. Each of the interactors is assigned different tasks by the

environment according to the definition of a story, such as rendering media objects and reporting the user responses during different periods of time.

4.1.3 Media objects

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

4.1.3.1 Abstract media objects

Traditional media objects are audiovisual objects, such as audio, video, text, and 2D/3D graphic objects. Some new standards like MPEG-4 have introduced new media objects that have a higher level of abstraction, e.g., face and body animation [22][29]. We go one step further: As its definition implies, a time-based media object here can be even more 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.

The definition and the range of abstract media objects are application-dependent. In a storytelling application, "behavior" is defined as a time-based object for all the interactors: Audio-visual devices, robots, lights and surrounding audio systems. All these devices have a capability to render "behavior".

Different interactors can render the same media object in a different way, but it should entertain the user with a similar experience that is predefined by the author. For example, at a certain moment in a program, both a robotic toy and an ambient light can render media object "happiness" from the program. The user's perception doesn't change, but the presentation of "happiness" is different.

The user may have different configurations of an environment. 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. For example, a content producer can specify "from now on for 10 seconds, performs the 'happiness' behavior on a robot". The author doesn't have to know whether there is a robot present in the environment or not, and if there is one present, how this robot will perform

the ‘happiness’. It solely depends on the configuration of the environment and the implementation of the robot.

4.1.3.2 Time dependencies

As introduced in chapter 2, many multimedia systems make a high-level distinction between time-dependent media objects (e.g. audio, video streams) and time-independent media objects (e.g. images, text, graphics). The reason to do so is that the system has to facilitate both media types with starting and stopping mechanisms; time-independent media objects are not intrinsically related to any points in time. The system doesn’t have to support this kind of media with mechanisms that can report media time and receive notification on whether a specific time point has been reached during playing these objects.

To distinguish between these types of media, the following assumption has been made. After the time-independent media objects have been started and before they are stopped, they will stay static, or they will be animated asynchronously. During playback, the time-independent media will not be synchronized with other objects, nor will they generate any time-based events. Furthermore, they can’t start and stop by themselves since there are no internal time dependencies defined. The playback of such media in a multimedia presentation has to be controlled by externally imposed time dependencies. This will be sometime troublesome in a distributed presentation environment.

Media objects are distributed to different self-contained interactors. If the time-dependencies have to be externally imposed on the object, the control information has to be dispatched separately along the network. This increases the complexity of media synchronization.

Storylines, feed-forward and feedback components in a story can be recognized as time-based media objects. The playback of these objects is related to an implicit timeline. To simplify the situation, a dimension of time is added to all of the media objects, that is, time-dependent and time-independent media objects are all time-based. They are encapsulated with internal time dependencies and thus the complexity of time control is hidden inside a media object.

Figure 4-2 shows the object-oriented model of an interactive story and the environment. To summarize, an interactive story application defines an interactive story and a desired environment. The story consists of many storylines and a definition of possible user interaction during the story. User

interaction can result in switching between storylines, or changes within a storyline. Dialogues make up the interaction. A dialogue is a linear conversation between the system and the user, which in turn consists of feed-forward from the media objects, and depends on the user's response and the feedback from the media objects. The environment may have many interactors. The interactors render the media objects. And finally, the story is rendered in an environment.

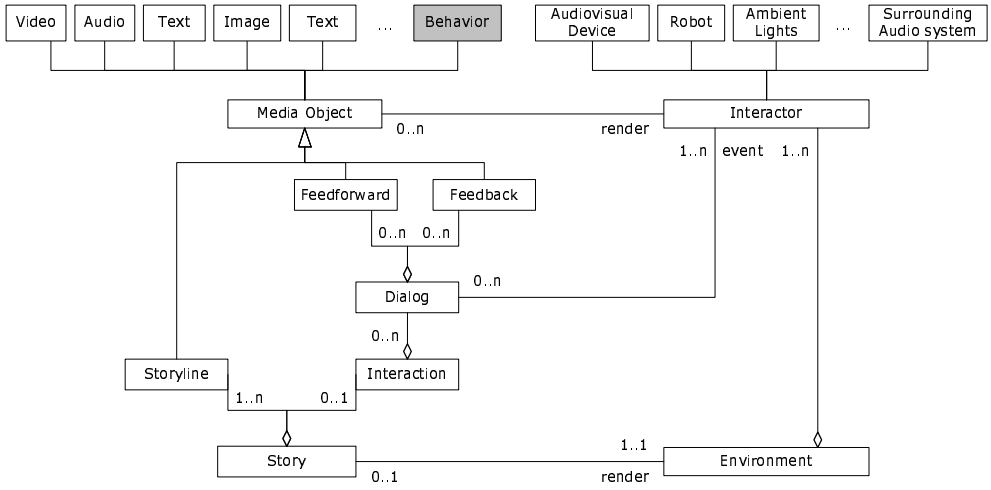


Figure 4-2.iStory object-oriented model

4.2 STORYML: XML-BASED DOCUMENTATION

4.2.1 Why XML

Based on the object-oriented story model, an XML based specification language, i.e., StoryML, is developed for authoring, serving, delivering and presenting an interactive story. XML stands for eXtensible Markup Language. Markup languages can be used to identify the document structures. The common markup languages currently in use are SGML and HTML. SGML (Standard Generalized Markup Language) is a standard system for defining and using document formats [18]. HTML (Hyper Text Markup Language) is a language used for hypertext linking, multimedia and displaying of simple documents on the Web [47].

XML is designed to provide an easy-to-write, easy-to-interpret, and easy-to-implement subset of SGML. It is not a fixed format like HTML. XML is a meta-language used to define other markup languages for structured documents. Structured documents are those that contain content stored

hierarchically, in a specified format. In this sense, HTML is just one of the SGML or XML applications. XML is designed so that a particular markup language, such as StoryML, meets the application needs more quickly, efficiently and logically.

4.2.2 StoryML elements

StoryML is developed based on the object-oriented model presented in Figure 4-2 . The StoryML elements are directly derived from the objects in this model. The syntax of StoryML documents is defined by the DTD, which can be found in [Appendix A. DTD (Data Type Definition) of StoryML]. An application example of StoryML can be found in [Appendix B. TOONS in StoryML].

4.2.2.1 The "id" attribute of a StoryML element

Each element of StoryML has an "id" attribute. This attribute uniquely identifies an element within a document. Its value is an XML identifier.

4.2.2.2 Attributes of a media object

In a StoryML document, the "storyline", "feedforward" and "feedback" elements define a media object. These media objects have the following common attributes:

src	The value of the src attribute is the URL of the media object
content	If the media object can not be retrieved from the network with its URL, the media data can be defined directly using this attribute. This attribute is often used for an abstract media object, e.g., a robotic behavior.
interactor	This attribute defines the "id" of a desired interactor on which this media object will be rendered.
type	This attribute specifies the type of the media object, which has fixed value of "audio", "video", "audiovisual", "text", "image" "graphics" and "behavior". The default value is "audiovisual". Specifying the media type of a storyline implies that, if the desired interactor doesn't exist in the environment or is not capable of rendering this storyline, the media object will be rendered by an alternative

interactor if suitable.

4.2.2.3 The "StoryML" element

The "StoryML" is the root element of a StoryML document.

Element attributes:

id	Defined in 4.2.2.1

Element content:

environment	Defined in 4.2.2.4
story	Defined in 4.2.2.6

The "StoryML" element may contain one interactive story and one desired environment.

4.2.2.4 The "environment" element

The "environment" element determines a desired configuration of the environment, in which the interactive story will play.

Element attribute:

id	Defined in 4.2.2.1

Element content:

interactor	Defined in 4.2.2.5

The "environment" element may contain many "interactors" which will be used for presenting the media objects.

4.2.2.5 The "interactor" element

The "interactor" element specifies a desired interaction agent for the interactive story.

Element attributes:

id	Defined in 4.2.2.1

type	This attribute indicates of the interactor's capability of media presentation. The "type" of an interactor also implies the possible user events which can be generated and abstracted by this interactor. The value of "type" is predefined: "audiovisual" for an audiovisual device, or "robot" for a robotic device. This can be extended in the future version of StoryML to support more types of interactors. The default value of "type" is "audiovisual".

Element content: The "interactor" element is an empty element.

4.2.2.6 The "story" element

The "story" element defines the storylines and the user interaction. All these storylines will be started at the same point in time. This point is then referred as time "0" on the implicit timeline. The maximum duration of these storylines implies the duration of the story. Operations on the story, such as starting, stopping, suspending, and resuming, are imposed to all these storylines, the implicit timeline, and the interaction elements as well.

Element attribute

id	Defined in 4.2.2.1
title	This attribute defines the story title, which can be shown by a player, e.g., to the title bar of a window, or on top of a video presentation at the beginning of the story.

Element content:

storyline	Defined in 4.2.2.7
interaction	Defined in 4.2.2.8

The "story" element may contain many "storyline" elements and one "interaction" element.

4.2.2.7 The "storyline" element

The "storyline" element defines a media object, normally an audiovisual media object, which defines a storyline for the "story" element.

Element attributes:

id	Defined in 4.2.2.1
src	Defined in 4.2.2.2
content	Defined in 4.2.2.2
interactor	Defined in 4.2.2.2
type	Defined in 4.2.2.2

Element content: The “storyline” element is an empty element.

4.2.2.8 The “interaction” element

All possible user interactions in an interactive story are specified in the “interaction” element.

Element attribute

id	Defined in 4.2.2.1

Element content:

Dialog	Defined in 4.2.2.9

The “interaction” element may contain many “dialog” elements

4.2.2.9 The “dialog” element

The “dialog” element specifies a linear session between the story and the user. The story starts the session by sending “feedforward” media objects to desired interactors, these interactors will then listen to the user responses. The user response will trigger an immediate “feedback”. This session will result in switching between storylines and or a change to a storyline

Element attribute

id	Defined in 4.2.2.1
begin	This attribute specifies the time in milliseconds for the explicit begin of a dialog. The time refers to a point on the implicit timeline which starts at time “0”. Right on the “begin” time, the dialog will start the “feedback” objects.

end	This attribute specifies the time in milliseconds for the explicit end of a dialog. The time refers to a point on the implicit timeline which starts at time "0". By the "end" time of the dialogue, all the media objects specified in the "dialog" element will be stopped and eliminated, and the interactor will no longer listen to the user events specified in this "dialog"
wait	This attribute specifies the time in milliseconds until when the dialogue will wait for the user responses from the interactors. After this time, the feedforward media objects will be stopped and eliminated, the interactors will no longer listen to the user events specified in this "dialog", and a default user response will be taken. If the default user response is not specified in this "dialog" element, the first "response" element will be treated as the default response. If this attribute is not specified, the "end" time will be taken as the "wait" time.
type	This attribute specifies what kind of result will happen to the storyline, whether an "immediate" or "delayed" result. The immediate result will happen right when the user responds. The delayed result will happen when the dialog is finished, that is, at the "end" of the dialog. The default value is "delayed".

Element content:

feedforward	Defined in 4.2.2.10
response	Defined in 4.2.2.11

The "dialog" element may contain many "feedforward" and "response" elements. In case no "response" element is specified for a "dialog", it can be used as a mechanism for rendering the "feedforward" media objects on the interactors during a certain period in time, without expecting user responses..

4.2.2.10 The "feedforward" element

The "feedforward" element specifies a media object, which will be rendered by an interactor as feed-forward information.

Element attributes:

id	Defined in 4.2.2.1
src	Defined in 4.2.2.2
content	Defined in 4.2.2.2
interactor	Defined in 4.2.2.2
type	Defined in 4.2.2.2

Element content: The “feedforward” element is an empty element.

4.2.2.11 The “response” element

The “dialog” element specifies a possible user event which can be expected form an interactor. Once this user event is received, it will trigger the feedback media objects to be rendered by desired interactors immediately, and will result in switching between storylines or a change to a storyline.

Element attribute

id	Defined in 4.2.2.1
interactor	This attribute specifies the “id” of an interactor, which will listen to certain user event during the “begin” and “wait” time of the dialog
event	This attribute specifies the expected user event from the interactor.
storyline	This attribute specifies the “id” of a storyline.
action	If the user event is received from the interactor, one of following actions can be taken, which is specified by this attribute: – “switchto”: switch to the storyline (specified by the “storyline” attribute) on the interactor (specified by the “interactor” attribute).

– “change”: a change will be made to the storyline.

The default value of this attribute is “switchto”.

changecontent — If the value of “action” attribute is “change”, this attribute specifies what kind of change will be made to the storyline.

default — This attribute specifies if a response will be used as a default response if the user doesn’t respond before the “wait” time of a dialog. The value if this attribute can be either “yes” or “no”, the default value is “no”.

Element content:

feedback — Defined in 4.2.2.12

The “response” element may contain many “feedback” elements.

4.2.2.12 The “feedback” element

The “feedback” element specifies a media object, which will be rendered by an interactor as feedback information.

Element attributes:

id — Defined in 4.2.2.1

src — Defined in 4.2.2.2

content — Defined in 4.2.2.2

interactor — Defined in 4.2.2.2

type — Defined in 4.2.2.2

Element content: The “feedback” element is an empty element.

5 PAC-ING THE STORYML PLAYER

In chapter 4, 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 a StoryML player.

The definition of appropriate software architecture is an important issue in user interface design. Without an adequate architectural framework, the construction of interactive system is hard to achieve, the resulting software is difficult to maintain and iterative refinement is made impossible. iStory, the StoryML player, is considered as a complex interactive system. The structure of such an interactive system is provided by its architecture.

A number of architectural models have been developed and progressively refined for the software design of interactive systems. These include the Seeheim model [33] revisited by the Arch model [45] which in turn has been refined in terms of the multi-agent paradigm.

5.1 AGENT-BASED SOFTWARE ARCHITECTURES

Agent-based models structure an interactive system as a collection of specialized computational components called agents that produce and react to events or stimuli. An agent has a state, possesses an expertise and is capable of initiating and reacting to events. Thus an agent is a unit of competence which operates in parallel and in coordination with other agents. Agents of an interactive system that communicate directly with the user are so-called "interactors". An interactor provides users with a perceptual representation of its internal state. It listens to the users, possesses an expertise which is convenient for them, and provides them with an adequate feedback [7].

An interactive system like iStory can take the following advantages of agent-based architectures [10].

- Agent-based models stress a highly parallel modular organization and distribute the state of the interaction among a collection of co-operating units. Modularity, parallelism and distribution are convenient mechanisms for supporting the interactive design of user interfaces, for implementing physically distributed applications, and for handling multi-thread dialogues.

– The agent-based models can easily be implemented in terms of object-oriented languages: an object class defines a category of agents where class operators and attributes respectively model the instruction set and the state of the agent, and where an event class denotes a method. Each object can be implemented as a thread in a multi-thread operation system, which enables the agent as an active object. The inheritance and aggregation mechanism provided by the object-oriented languages can be usefully exploited to modify a user interface without changing the existing function core.

Many agent-based models have been developed along the lines of the object-oriented and the event processing paradigms. MVC and PAC are the most popular and often used ones [15].

5.1.1 MVC (Model-View-Controller)

The MVC model 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. The model is independent of specific output representations or input behavior. 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. Events are translated to serve requests for the view or the model. The user interacts with the agent solely through controllers.

The separation of the model from the view and controller components allows multiple views of the same model. If the user changes the model via the controller of one view, other views dependent on this data should reflect the changes. The views in turn retrieve new data from the model and update the displayed information.

5.1.2 PAC (Presentation-Abstraction-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 communicating 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 [7] (cf. Figure 5-1).

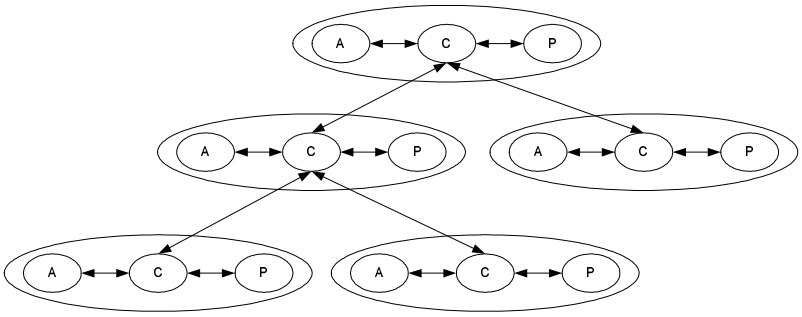


Figure 5-1.Hierarchy of PAC agents

An interactive application can be structured as a tree-like hierarchy of PAC agents. The top level PAC agent provides the function core of the system; bottom-level agents represent self-contained semantic concepts on which users of the system can act. Intermediate-level agents represent either combination of, or relationships between, lower-level agents.

5.1.3 Comparing PAC with MVC

The PAC based architecture takes a different approach to decoupling the user interface aspects of an agent from its functional core. Its abstraction component corresponds to the model component in the MVC model, and its presentation component corresponds to the view and controller components in the MVC model. . The communication between the abstraction and presentation components in the PAC model is decoupled by the control component. The interaction between presentation and abstraction components is not limited to just calling an update procedure, as it is within the MVC model.

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

– The iStory system 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. The MVC architecture lacks this. It is crucial to have such a mechanism for a distributed application like iStory. 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 iStory 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 iStory system 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.

5.2 EXTENDING PAC FOR TIME-BASED MEDIA

In a PAC system, the control components are the communication mediators between the abstraction and presentation parts of an agent, and between different PAC agents. The quality of the control component implementations is therefore crucial to an effective collaboration between agents and for the overall quality of the system architecture. The individual roles of components should be strongly separated from each other. The implementation of these roles should not depend on specific details of other agents.

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 [15].

This may trouble the use of the PAC architecture for the iStory system. Time-based media objects are distributed over the PAC hierarchy in real-time. These objects are often video, audio streams, which are considered as mass data. According to the PAC paradigm, the system has to stream these objects along the network of control components, whereas other important real-time controlling messages or events might be stuck in the PAC hierarchy.

To overcome this potential pitfall, iStory extends the abstraction component. For time-based 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 (cf. Figure 5-2).

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.

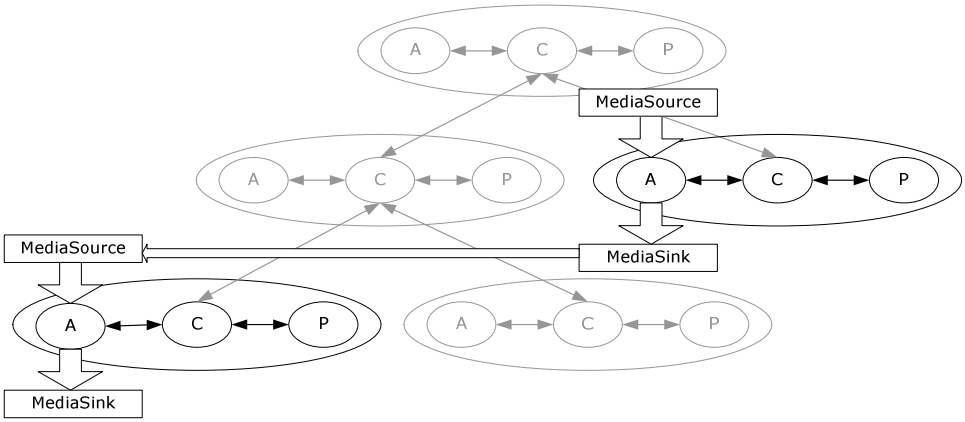


Figure 5-2.PAC for time-based media

5.3 iSTORY ARCHITECURE

Figure 5-3 shows the semantic hierarchical structure of the TOONS application.

The top-level iStory agent provides the functional core of the system. Most other agents depend or operate on this core. It communicates with the content server, provides access to the content and its metadata, parses the application logic from the content document, and presents the interactive story to the user.

The content portal establishes the connection to content servers provides the system with time-based content. In many cases, particularly when presenting a media stream that resides on the network, the presentation of the media stream cannot begin immediately. The time it takes before presentation can begin is referred to as the start latency. The content pre-fetcher overcomes the start latency by pre-fetching certain amount of data, and ensures that the media objects are prepared to start at certain moments.

The interactive stories are documented in StoryML. An XML parser can first parse the document into DOM (Document Object Model) objects then in turn the StoryML parser translates them into internal representations.

At the bottom level are several agents which reside in, and respectively indicate different physical interface devices. These devices are often equipped with embedded processors, memory, and possibly some input and output accessories. The audiovisual device shows the live audiovisual objects which present the main content of the story. This can be a PC or a TV set. Normally the story or program occupies the full screen and the audio channels. The robot renders extra multi-modality information or some robotic behaviors as feedback and feed-forward to the user and gathers user responses to the story. The ambient light renders the information about mood from the story or program. Users can adjust the brightness of the light as mood input as well. More physical agents can be involved into iStory architecture at this level.

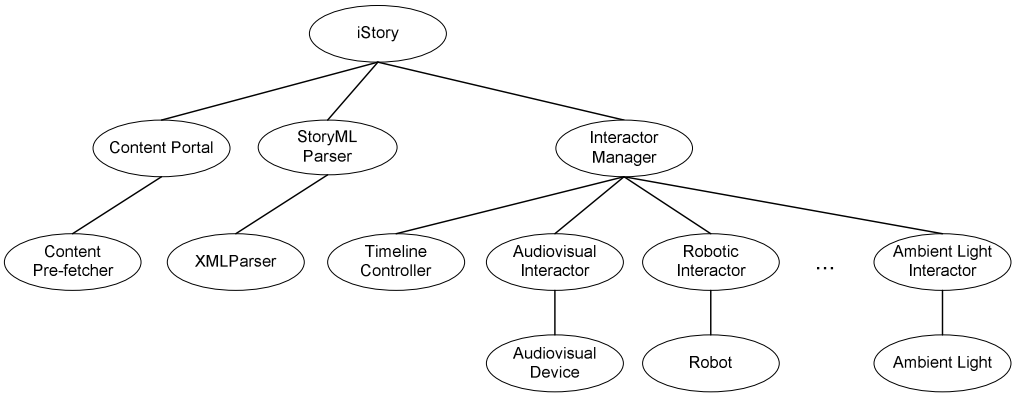


Figure 5-3.PAC based iStory architecture

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:

An intermediate virtual interactor is connected to each physical agent, e.g., a robot, as its software counterpart. With this layer of virtual interactor(s), 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. For example, if the robot's processor is not powerful enough to decode an audio stream, its software interactor in a PC will do it and then send the audio signal directly to the robot's audio output.

- 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 interfaces 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. If the user interaction on one of the software interactors will result in changes on the others, the interactor manager ensures those software interactors will be notified right away or register these changes to the timeline controller, depending on the temporal nature of these changes.

The interactor manager also maintains a timeline controller, which plays an important role in synchronizing user interaction with the story. The dialogs between the user and the story are always registered to and initialized by the timeline controller. If the user response results in a later change to the storyline, this change will be also registered to the timeline.

6 ISTORY MEDIA AND INTERACTION SYNCHRONIZATION

PAC-based architecture paves the basic infrastructure for playing an interactive story on multiple interactors in a distributed presentation environment. The content distributed to these interactors has to be synchronized according to their nature or application semantics. A time dependent change-propagation mechanism has to 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.

This chapter presents the synchronization mechanisms developed in iStory system, in the context of a multimedia synchronization reference model.

6.1 MEDIA SYNCHRONIZATION

6.1.1 Synchronization reference model

A layered reference model for multimedia synchronization is introduced by [26] and further developed in [3]. The model provides four layers of abstraction through which a multimedia application can access synchronization services. Each layer implements synchronization mechanisms which are provided by an appropriate interface. These interfaces can be used to specify or enforce the temporal relationships. Each interface defines services, which can be used by an application directly, or by the next higher layer to implement an interface.

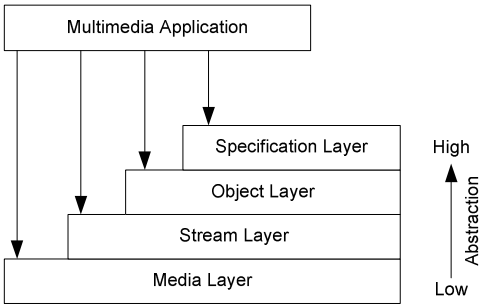


Figure 6-1.Synchronization reference model [3]

The media layer provides a multimedia application with the capability to perform operations on single, continuous media stream. A media stream is

considered to be made up of a sequence of logical data units (LDUs), such as audio samples or video frames, which are to be played back at a specified rate. At the media layer the application is responsible for the LDUs of a stream being played back at correct intervals, thus enforcing intra-media synchronization.

The stteam layer operates on continuous media streams, as well as groups of media streams. In a group all streams are presented by using mechanisms for inter-stream synchronization. Grouped streams are played back in parallel, controlled by a common clock or time-base [37]. The main responsibilities of an application are the grouping, starting and stopping of media streams. While the application is not required to operate in a real-time environment, stream playback is assumed to occur under real-time conditions.

The object layer provides a unified abstraction for any type of media, time-dependent (e.g., audio, video streams) or time-independent (e.g., graphics, text objects). The abstraction offered to the application is that of a complete, synchronized presentation. This layer takes a synchronization specification as input and is responsible for the correct schedule of the overall presentation.

The specification layer is responsible for producing a synchronization specification representing a certain document. The layer contains any tools, such as authoring systems, used to create or modify synchronization specifications. Rather than postulating concrete systems the specification layer is characterized by the various temporal models used to define synchronization relationships between media objects in a presentation.

6.1.2 Synchronization specification methods

According to [3], synchronization speciation methods can be classified into the following main categories:

- Interval-based specifications, which allow the specification of temporal relations between the time intervals of the presentation of media objects. Temporal intervals are defined by their starting points and ending points. In [1], the author introduces 13 basic relations that can exist between temporal intervals, namely such as *before*, *meets*, *overlaps*, *during*, *equals* and so on.

- Axes-based specifications, which relate presentation events to axes that are shared by the objects of the presentation. A common model of axes-based specifications is the *timeline*. Based on a global timer, the

synchronization is described by attaching all objects, independently of each other, to a time axis. Removing one object does not affect the synchronization of the other objects.

– Control flow-based specifications, in which at given synchronization points, the flow of the presentations is synchronized. Many methods belong to this category, such as hierarchical specification, reference points, and timed Petri nets [3] . One of the recent examples is the SMIL time model, in which sequential and parallel playback of media items, respectively, represented by synchronization elements <seq> and <par> [48][49]. Media objects are regarded as a tree consisting of nodes which denote serial or parallel presentation of the outgoing sub-trees, which forms a hierarchical control flow.

– Event-based specifications, in which events in the presentation of media trigger presentation actions. Typical presentation actions are start, stop and prepare a presentation [37]. The events that initiate presentation actions may be external (e.g. generated by a timeline) or internal to the presentation generated by a time-dependent media object that reaches a specific LDU.

6.1.3 Interaction specification

Time-based content interaction involves the user input as a parameter of media presentation, with which the user can manipulate the media objects, or influence later presentation of other objects in the content. The user interaction has a temporal nature which is related to certain media objects or the overall content. At the specification layer, the interaction therefore has to be synchronized with media presentation. This is often done by attaching input sensors or hyperlinks to media objects in the content document.

In SMIL, in addition to the "A" element inherited from HTML allows associating a link with a complete media object, the "ANCHOR" element allows associating a link destination to not only spatial, but also temporal subparts of a media object [48][49].

In a BIFS scene of MPEG-4, sensor nodes can be attached to media objects, detect events in their environment and fire events. For instance, a TouchSensor detects a click of a mouse, a ProximitySensor detects that the user entered a region of the space. Sensor nodes are classified in three categories: nodes usable for 2D and 3D sensors, 2D specific nodes, and 3D

specific nodes. Interpolator, Sensor and ROUTE statements enable the design of interactive scenes.

6.2 iSTORY SYNCHRONIZATION

One of the key characteristics of the iStory system is related to media and interaction synchronization issues. In this section the system is classified and described according to the above mentioned synchronization reference model. Figure 6-2 presents a summary of how various elements of the iStory system related to the reference model.

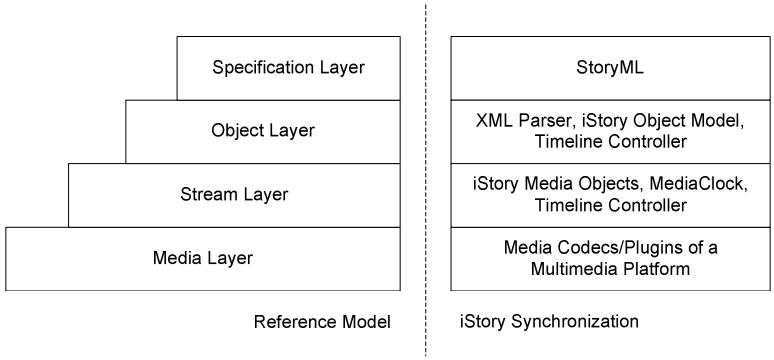


Figure 6-2.How iStory related to the synchronization reference model

6.2.1 iStory specification layer

iStory does not provide services explicitly belonging to the specification layer, only the interface between the specification and object layer is defined. The specification layer is responsible for producing a StoryML document, which contains the synchronization specification for a presentation. The methods and tools for producing the document remain a variety of options such as basic text editors, visual authoring tools or dynamic generation. The StoryML document acts as the input to the object layer.

In StoryML, media objects and interaction dialogues refer to an implicit timeline by specifying their starting and stopping point in time (cf. Chapter 4). The metaphor behind it can be easily understood by comparing with the conceptual model of the interactive story (cf. “Conceptual model of TOONS” in Chapter 3). 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.

6.2.2 iStory object layer

6.2.2.1 Parsing the StoryML to internal object models

A crucial part of iStory 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. iStory implements a global timeline controller (cf. Figure 5-3) for synchronizing media objects and interaction, which might 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.

6.2.2.2 Event based interaction synchronization

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. Unlike MPEG-4 or SMIL, 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 in iStory system.

6.2.3 iStory stream layer

6.2.3.1 Ticktacking media objects

In iStory, a media object is defined as an entity which can be rendered by an interactor. Audio or video streams, text, graphics, images, robotic behaviors, and a composite object, are all media objects (cf. Chapter 4).

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.

A MediaClock uses a TimeBase to keep track of the passage of time while a media stream is being presented. The only information that a TimeBase provides is its current time, which is referred to as the time-base time. The time-base time cannot be stopped or reset. Time-base time is often based on the system clock and viewed as a real world clock.

Even these 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.

6.2.3.2 MediaClock based media synchronization

The timeline controller also implements a MediaClock, which the system keeps as the master MediaClock. Media objects are registered to the timeline controller with their starting and stopping time. Once a media object is started by the timeline controller, its MediaClock is synchronized with, or in other words, controlled by the master MediaClock. If the media objects are not fast enough, then reduce the quality (e.g. drop frames in a video stream) to catch up if necessary (cf. Figure 6-3).

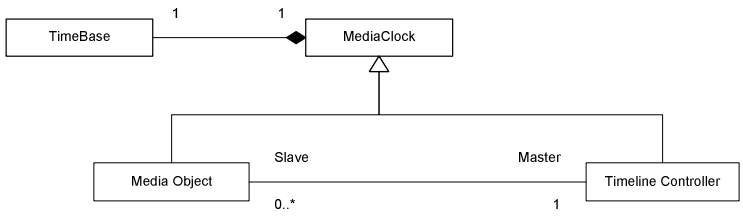


Figure 6-3.Timeline based media synchronization

6.2.4 iStory media layer

The intra-media synchronization issues are not in the interests of this project. Many multimedia platforms, such as JMF (Java Media Platform), have various codecs or plugins for many different media formats providing the media layer services. iStory leaves the design and implementation of this layer to these dedicated platforms.

7 ISTORY IMPLEMENTATION

One day in the year 2001. 6:00pm in the afternoon. Xiaoxiao, a 6-year old boy, is watching a storytelling program TOONS showing on a PC screen, together with his little robot Tony. It seems that Tony is not so interested – Tony has a light on his head but now it is off. He is sleeping.

"Hi, wake up." Xiaoxiao pats Tony's side. He wants to watch the program together with Tony. But Tony is a bit annoyed and reacts with a descending arpeggio.

In the story, a girl comes to the front gate of a castle. Suddenly, one of the rooms in the castle bursts with terrifying sounds. Xiaoxiao noticed that Tony is woken up by the sounds and his head light is now dimly on.

"Finally you wake up." Xiaoxiao jogs Tony. This time Tony is happy to be touched and plays an ascending arpeggio, but still moves away a bit from Xiaoxiao.

The girl wants to know what's happening in the castle. She browses through the rooms for some time and now she is standing in front of two doors. "Which door should I enter? They all look the same...The left? The right? Hmm, I am not sure...Tony, maybe you can help me..."

The story catches Tony's attention. Tony was moving back and forth, flashing his head light., now he is standing there with his head light brightly on.

"Hi, Tony, it seems that you can help her. Hmm...The left door looks nicer". Xiaoxiao taps the left side of Tony. Tony reacts with a beep.

In the story, the left door opens for the girl. Behind the door there is a beautiful garden with colorful trees and puffy bushes.

After a while, Tony is getting tired, his head light becomes dim again.

...

For demonstration purpose, the iStory system 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 the content portal, StoryML

parser, timeline controller and interactor manager. All these components of both sides are based on Java [40][43] technology. Figure 7-1 shows the system architecture of the experimental implementation.

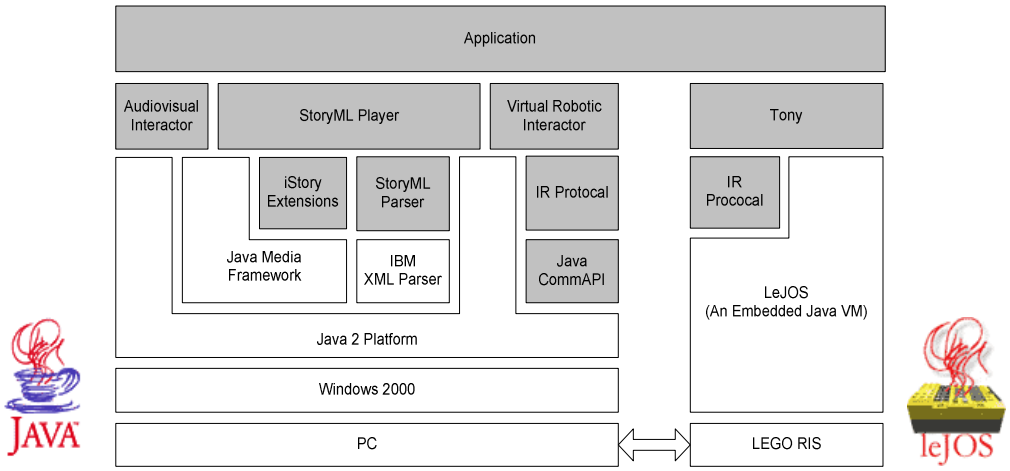


Figure 7-1.iStory experimental system architecture

7.1 AT THE PC SIDE

7.1.1 Java 2 Platform

The java 2 platform [37] provides the operating system services for iStory at the PC side. It serves as an abstraction between the upper layer building bocks in the architecture and the operation system, i.e., Windows 2000 in this implementation. It provides cross-platform portability, a secure runtime environment, and networking and connectivity APIs.

In addition to the programming language and object-oriented model, iStory uses the following services provided by the Java 2 platform:

- The Java build-in multi-thread supports for concurrent processing, which serves as the infrastructure for interactive agents. Agents are implemented as one Java thread and Java events are used for effective and light-weight communication between agents.

- Threads are also used to perform a variety of asynchronous tasks. The most important scenario is the enforcement of the playback and timeline schedule.

– The JFC/Swing user interface toolkit [40] is used to construct the presentation component of the agents at the PC side, such as the audiovisual interactor, the StoryML player, and so on.

7.1.2 Java Media Framework and iStory extensions

The Java Media Framework API (JMF) [39] enables audio, video and other time-based media to be added to Java applications and applets. JMF provides iStory the means for controlling individual media streams in a way that is independent from delivery mechanisms, transport protocols, media types and encoding formats. The iStory system uses JMF 2.1.1 mechanisms to synchronize media and deal with the timing issues.

Figure 7-2 shows the key java interfaces defined by JMF and iStory extensions. The Clock interface defines the basic timing and synchronization for the control of the media object playback in the JMF. A Clock has a TimeBase object which presents the flow of the real world time, independent of any other media objects. The media time of a media object, which has a Clock, represents the current position in time in the media stream of the object. The iStory system implements the interface of the Clock (namely MediaClock in the iStory system) and TimeBase for all media objects, including robotic behavior.

The Controller interface extends Clock to provide methods for managing the system resources and preloading data, and a listening mechanism that allows the Player to receive notification of media events. It defines a state model for controlling the transition of a media object through various resource allocation states. The interface also defines methods for registering event listeners, which will be notified through events of state changes or other occurrences in the Controller. The Duration provides a method to determine the duration of the media being played.

The Player supports standardized user control and relaxes some of the operational restrictions imposed by the Clock. To synchronize the player with other media players, all Players must use the same TimeBase. The Player's user interface can include both a visual component as well as a control-panel component. A Player has a DataSource, which represents the location and delivery protocol associated with a specific media object. The iStory MediaSource extents the DataSource for the abstraction component of a PAC agent which requires media inputs from other agents.

A JMF Processor is a Player that takes a DataSource as input, performs some user-defined processing on the media data, and then outputs the processed media data. A Processor can send the output data to a presentation component of an agent or to a DataSource. While the processing performed by a Player is predefined, a Processor allows the application developer to define the type of processing that is applied to the media data. This enables the application the creation of effects, and the mixing and composing of media data in real-time. If the data is sent to a DataSource they can be used as the input to another Player or Processor. This kind of DataSource is extended as a MediaSink in the iStory System. A pipeline can be built between a MediaSink and a MediaSource in the PAC-based iStory architecture to provide direct media transportation between PAC agents, using real-time streaming protocols.

An iStory Media Object Processor implements the interface of a JMF Processor, which can be used as a Player. It has a MediaSource to provide direct visual output to the presentation component of a PAC agent, or as a Processor to perform certain media processing and generate a MediaSink.

The iStory Timeline Controller implements the interface of the JMF Controller, which provides methods for an iStory Media Object to register as a TimeEvent Listener, which extents the Media Object by defining a point in time and a behavior of the Media Object. The Timeline Controller will generate a TimeEvent at the registered moment and notify all the TimeEvent Listeners, which are associated with this TimeEvent, to perform certain behavior.

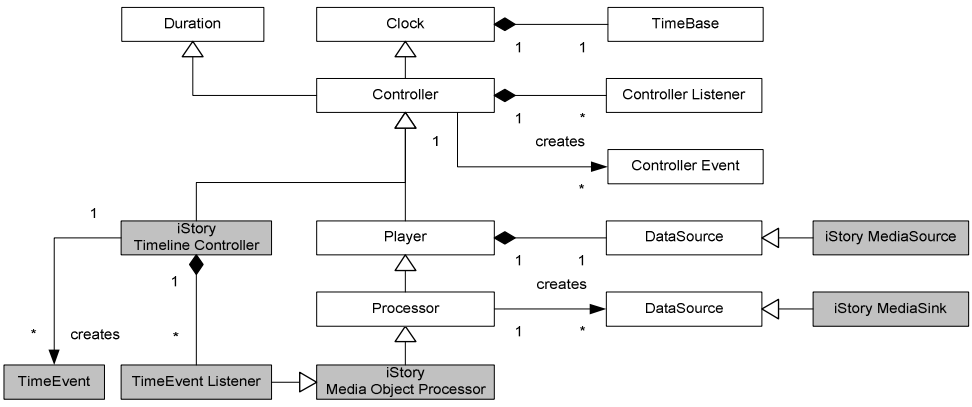


Figure 7-2.JMF interface hierarchy and iStory extensions

7.1.3 IBM XML parser

The iStory system uses the IBM XML Suite of Java Beans [17] to parse the StoryML to DOM (Document Object Model) objects, which allows iStory to access and update the content, structure and style of StoryML documents. It provides a rich set of functionalities to perform various XML related operations like viewing, searching, editing, or processing XML documents. The XML Bean Suite includes the following beans[4]:

- XMLCore beans are non-visual beans which are used by other beans. Primarily they provide a facility to convert an XML file to a DOM document or vice versa. Other beans in this suite operate on a DOM document rather than operating directly on XML files.

- XMLViewer beans are a set of generic viewers, which can be used to view any XML data, irrespective of the Document Type Definition (DTD) that it may follow.

- XMLEditor beans are designed to construct XML editors for editing valid XML data. These beans directly correspond to the various constructs of a Document Type Declaration (DTD), like elements, attributes, operators, etc.

- XMLProcessing beans provide a set of beans which can perform various processes like searching, filtering, tokenizing DOM nodes into strings etc. All these beans are non-visual beans and can be used with any GUI interface.

- XMLConvenience beans are pre-wired bean idioms for most common wiring scenarios in which XML beans can be used with AWT (Abstract Window Toolkit).

7.1.4 StoryML parser

The StoryML parser is built on the base of the IBM XML Suite. A DOMGenerator bean parses a StoryML document to a DOM document, which in turn is used as input for an XMLTokenizer bean.

The XMLTokenizer bean traverses the entire tree under the DOM Document and breaks it up into String tokens. Each time a DOM Node is encountered,

[4] The following descriptions about the IBM XML Suite of Java beans are edited from its online document.

it fires document related events like the start and end of elements/attributes/text, etc. The order of these events matches the order of information in the StoryML document. These events are used for the StoryML parser to construct the internal representation of the StoryML objects.

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

7.2 AT THE ROBOT SIDE

7.2.1 LEGO RIS

The robot used for the demonstration is assembled using the LEGO Mindstorms Robotics Invention System (RIS) [24]. RIS includes two motors, two touch sensors, one light sensor, more than 700 LEGO bricks, and a programmable brick RCX. The RCX (Robotic Command Explorer) contains a microcontroller, a Hitachi H8/3292, with a H8/300 processor, 16KB of ROM and 32 Kb of RAM. The ROM contains low-level routines for the motors and sensors. With an infrared tower connected to a serial port of a PC, a wireless infrared connection can be build between a RCX and the PC (cf. Figure 7-3).

Through the H8/3292 based device controllers the control program accesses RCX input/output devices like buttons, a speaker, and a LCD display. Furthermore, sensors like a touch sensor or a temperature sensor can be connected to the RCX input ports providing sensor input to the control program and the control program can activate actuators like motors connected to the RCX output ports. Stimuli from the environment can be registered by a sensor, e.g. a touch sensor, and transformed into input values for the control program. The resulting response can be accomplished by the control program through values output to actuators, e.g. a motor.

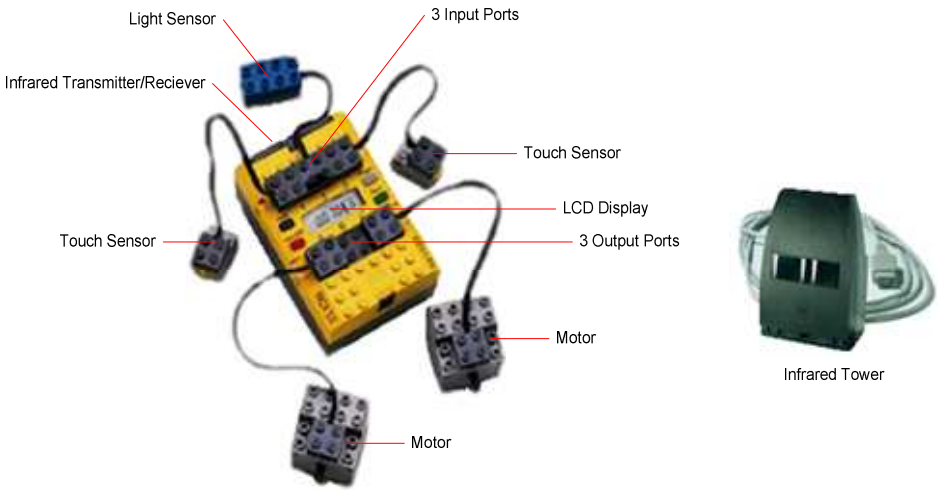


Figure 7-3.Some components of RIS

The development of control programs takes place on a host computer (e.g., a PC). The control program is downloaded to the RCX through a serial communication link provided by the infrared transmitter/receiver. The control program can be developed in different programming environments offered by LEGO or in other alternative programming environments.

7.2.2 LeJOS – an embedded Java virtual machine

There are two categories of alternate programming environments for the RCX. The first uses the default firmware on the RCX and provides alternate programming environments on the desktop PC. The second class of alternate programming environment uses replacement firmware and also provides a set of PC-side tools for programming. LeJOS falls into this category. LeJOS was created by Jose Solorzano, a Java developer and LEGO robot enthusiast [36].

Only a small subset of the Java Virtual Machine and APIs can be implemented on such a small device like the RCX. LeJOS is indeed a small Java Virtual Machine that is downloaded to the RCX to replace the standard firmware from LEGO. The current footprint of LeJOS in the RCX is 17 Kb.

Java programs are written and byte-compiled on the PC and downloaded to the RCX. 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. Furthermore, it has SensorListener and ButtonListener for the communication between several RCX-Bricks or between RCX and PC. It

also includes substantial additional functionality, including support for threads, floating point, string constants and multi-program downloading.

7.2.3 Tony

Figure 7-4 shows the robot, named Tony, used in the iStory system. Tony is equipped with four touch sensors. The left and front touch sensors are connected to the same input port on the RCX brick, while the right and back touch sensor occupy another input port. Depending on the context of the interaction and the status, Tony can distinguish which sensor is being touched.

A light on Tony's head is connected to one of the output ports on the RCX. With this light, the robot shows its attention at different levels with a different level of brightness. Tony has two wheels that are driven by two motors connected to another output port, so that it can move back and forth.

The speaker in the RCX can play ascending and descending arpeggio, which indicates respectively an annoyed or happy mood.

Tony is programmed in Java and simply can perform four different behaviors: it can be at sleep, relaxed, in attention, and tired. The LCD display on the RCX shows the state of the behavior that the robot is currently performing. Table 7-1 shows how Tony will execute different behaviors and possible interactions during a performance.

A request can be send from another device, for example, a PC, which has an infrared tower connected to its serial port, to Tony's infrared receiver. The term "request", instead of a "command", indicates that Tony *doesn't have* to perform the requested behavior – Tony is independent. Whether a behavior will be exhibited depends on its current mood and interests, and on the quality of the request (the quality of infrared connection, bad connection will make the request unclear for Tony).

Only when Tony is performing "in attention" and any 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.

Tony is presented to a StoryML show in the iStory system as an interface agent, and as a companion for the user. It watches the 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.

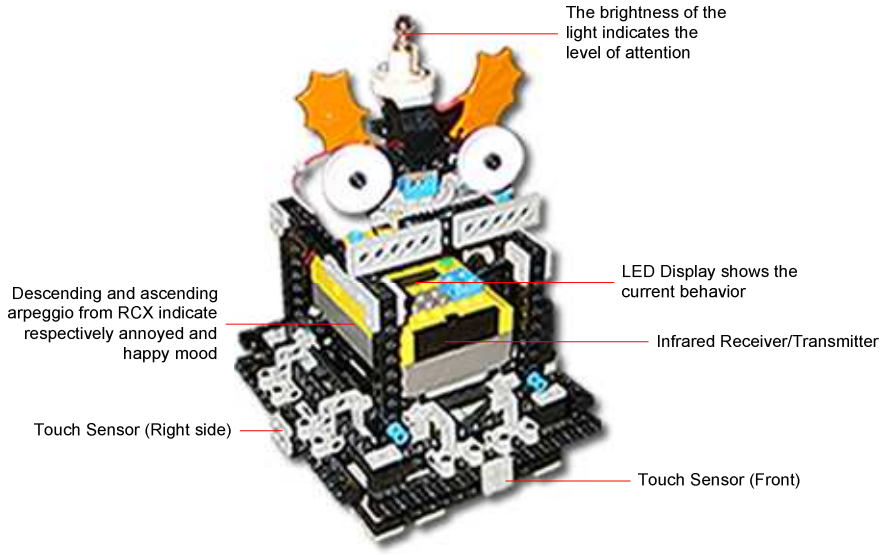


Figure 7-4.Tony [5]

Table 7-1.Tony's behaviors and possible interactons

Behavior		Sleeping	Relaxed	In Attention	Tired
Performance	Light	Off	Dim	Flashing for a while, then bright	Dim
	Sound	Quiet	Quiet	Ascending arpeggio then quiet	Quiet
	Movement	At rest	At rest	Move forth and back once, then at rest	At rest
Reactions if touched	Sound	Descending arpeggio	Ascending arpeggio	Beep	Descending arpeggio
	Movement	_____	Moving away	_____	_____

[5] Tony is based on a robot assembled by Christoph Bartneck.

	Events sent out			Which sensor is touched	

7.3 COMMUNICATION

In order to enable the bidirectional communication between the PC and Tony through the infrared connection, an IR Protocol agent is build at both sides based on the LeJOS and Java Communications API version 2.0 [38]. Java Communication API is a standard extension API that enables Java applications to talk to serial and parallel data ports on a computer. This version of the Java Communications API contains support for RS232 serial ports and IEEE 1284 parallel ports.

A Virtual Robotic Interactor is created at the PC side as Tony's software counterpart. This Virtual Interactor keeps verifying every 2 seconds whether Tony is present in the environment by sending a handshaking message. If Tony is not found, the Virtual Robotic Interactor will be visible on the screen with a standard graphic user interface, with which the user can interact by using standard input devices such as a keyboard or a mouse. If Tony is present in the environment, 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 the Tony's RCX through the IR tower. A supporting class was created to assist the IR Protocol agent in the asynchronous receipt of messages between the PC and the RCX.

The infrared communication between a PC and a RCX is originally designed for sending commands and data from the PC to the RCX. After sending out a message, the IR tower will wait for the verification from the RCX for only 4 seconds to save battery power. These 4 seconds are the only chance for the RCX to send messages back to the IR tower. To ensure continuous bidirectional communication between the IR tower and the RCX, the IR Protocol agent keeps the IR tower alive by sending out a NOP (No-Operation instruction) every 4 seconds.

7.4 SOME TECHNICAL DIFFICULTIES

7.4.1 Overlay graphics on video

One of the key requirements for showing an interactive program on a screen is to overlay GUI components, graphics and text over the video using transparency. This can be done in Java by JFC/Swing components. JFC/Swing components are light weight components. JMF by default uses heavy weight components for the visual component of a video stream. Heavy weight components are used to JMF's advantage as they permit using native rendering methods for higher frame rate video. However, the lightweight widgets are always covered by the heavyweight component of the video when the alpha channel is set, which makes the overlay problematic.

JMF 2.1.1 comes with several different video renderers. Although the default video renderer uses heavy weight components, it is possible to force a player to render using light weight components.

But in practice, the author found out that this solution is implementation dependent since it may not work with all players. Furthermore, although a transparent graphics, especially an animation, can be put on a light weight video component, the performance of video playback is badly reduced.

One of the possible solutions is to use Java 2D [41]. With Java 2D, It is possible to draw graphics and text into a Java 2D BufferedImage directly, when the video bits are about to be rendered. This functionality can be expected in future versions of JMF.

There is another alternative way to place a transparent component over the video. Usually, some dedicated platforms, such as Philips TriMedia set-top box, have a graphics processor which supports colors with transparency. This has been specified in the GUI API of DVB Java platforms as a TV extension [34].

In this project, this problem remains unsolved. To work around it, all the needed overlay components were merged into video streams. The problems with this method are that each light component has to be presented as a heavy video stream and that this limits the number of this kind of components. Direct manipulation on these light components is impossible.

7.4.2 Start latency

In JMF, starting synchronous playback is based on JMF knowing the Start Latency of each media object. That is, JMF calculates the maximum time it will take to start all involved threads in order to determine a common instant when all the threads can be started.

To determine how much time is required to start a media object, one can call the getStartLatency method of the media object. For media objects that have a variable start latency, the return value of getStartLatency represents the maximum possible start latency. But for some media types, getStartLatency might return LATENCY_UNKNOWN.

The iStory system often uses these media objects as feedback and feed-forward information. It means that these media objects have to be partly pre-fetched for immediate start-up. For such objects the start latency must be guessed. This introduces uncertainty in synchronously starting media streams.

7.4.3 Seamless video stream switching

The iStory system presents the storylines using MPEG-1 or MPEG-2 streams. One of the important requirements is to switch between these storylines seamlessly.

MPEG uses three different frame types; I, P and B. I frames are self-contained and independent, while P and B frames contain mostly frame differences. Jumping into a bit stream at a P or B frame means that the I frame to which the P or B frame refers is missing. The result is garbage until the next I frame. Although the garbage frames could be hidden with a black screen, the user will suffer flicking screen during stream switching.

Furthermore, the amount of time required to send MPEG frames varies. Because video frames occur at regular intervals, MPEG decoders use a buffer to reconcile the differences between input and output data rates. MPEG encoders must carefully construct bit streams to prevent decoder buffer over/underflow. Switching between different bit streams together can cause buffer management problems.

Seamless switching can be done with MPEG streams by first decoding to video, then switching the video and finally re-encoding the switched video. This is seamless and frame accurate, but has two drawbacks: high cost and reduced picture quality. Encoders are expensive, and each time the stream is

decoded and re-encoded, additional coding artifacts are produced. Directly switching the bit stream would be a better solution.

The focus of this project is not to solve this problem. So the iStory system takes another approach. As the first method mentioned above, the iStory system first decodes MPEG stream to video. This is done with the facilities provided by the JMF Players and results in visual components containing these video streams. These visual components are organized into a presentation window using Java AWT CardLayout manager. The presentation window is the container of these visual components. A CardLayout object is a layout manager for a container. It treats each component in the container as a card. Only one card is visible at a time and the container acts as a stack of cards. Switching MPEG streams is then simply done by bringing a visual component to the front in the presentation window using CardLayout manager, with DoubleBuffer enabled to achieve faster switching.

7.4.4 Programming and debugging RCX

Memory really is in short supply on the RCX. After all, one has only got 32KB of RAM to play around with, of which the LeJOS firmware takes up about 17KB. Furthermore, LeJOS doesn't actually implement garbage collection. If objects are created in a loop, or in response to sensor events, then it is probably going to run out of memory sooner or later. Discarded objects will never be collected as garbage.

Debugging on a small device like the RCX is difficult. One of the problems is that the development cycle is awkward – The program is coded and compiled on one machine (i.e. a PC) and tested on another (i.e. the RCX). LeJOS appears to include an emulator that can be used to test the code, but most of the functions of a Robot are related to its robotic input and output, which can hardly be simulated on a PC.

To debug the LeJOS program on a RCX is rather tricky. One of the methods is to put the debug information on the LCD screen. Although only five characters can be shown on it, it is still enough to be a useful debug tool. Another possibility is to play sounds. It is useful to signal when the program makes certain decisions or when it detects specific sensor conditions.

LeJOS can also display an uncaught exception on the LCD screen. The exception is displayed as AAAA B, where AAAA are the method signature numbers and B is the exception class index number. With a signature number

and class index check list, it is possible to figure out what exception was thrown and which method threw it.

8 CONCLUSIONS AND FUTURE DEVELOPMENTS

In chapter 1, the focus of the work presented in this report has been brought forward in a question: **how to structure the system and content to support distributed interfaces for time-based media applications**?

First this question was addressed by evaluating and using existing content documentation and system architecture technologies. By using user-centered approaches, these technologies were analyzed and examined in the context of user requirements. The author concluded that these technologies could hardly satisfy these user requirements and that new approaches had to be developed.

8.1 HOW ISTORY SATISFIED REQUIREMENTS

The iStory system was developed as a basic framework and a starting point for enabling interactive time-based media content to be presented in a networked environment, e.g., the user's home. The design and implementation of this experimental system, i.e., the iStory system was presented. The goal of the design and implementation is not a system that will be used for production purposes.

The iStory system structures the interactive story in StoryML, an XML-based content documentation language. To play a StoryML document, a StoryML player was developed with an extended PAC-based architecture. MediaClock and timeline based mechanisms enable the media and interaction synchronization in this architecture. These technologies together satisfied the requirements which were presented in section 3.2.

8.1.1 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.

8.1.2 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 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.

These physical devices might or might not always be available in the environment of the user. Since the configuration of the user environment is dynamic, an application can not assume a static, stable or pre-defined configuration of available physical devices for interaction. Many physical devices can be present in this type of end-user environment. 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.

8.1.3 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 TimelineController, which monitors and manages the synchronization. This agent implements a MediaClock as the basic clock, to which all the media object's MediaClocks are synchronized. Time-based events are registered to and triggered by the timeline at certain moments to start, suspend, resume and stop a dialog between the user and the content.

8.2 STRENGTHS

Many strengths of the iStory system come with open technologies. The iStory system is based on XML and Java technologies. A combination of XML and Java is natural: "XML is the portable data and Java technology is the portable, maintainable code" [44]. XML is the preferred technology in many

information-transfer scenarios because of its ability to encode information in a way that is easy to read, process, and generate. XML gives Java something to do [4]. Java provides XML with the capabilities of platform-independent data processing. Together, XML and Java technologies provide the iStory system with strengths of simplicity, portability, and flexibility.

JMF decouples both media types and network protocols from applications. Integrating support for new media types and protocols into the iStory system is straightforward. JMF 2.0 plug-in architecture makes it easy to add new codecs for emerging standards, such as MPEG-4 audio/video codecs. The Java programming environment allows developing Java plug-ins that will work on any Java platform, or plug-ins that make optimal use of the native hardware/software audio and video facilities.

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.

8.3 WEAKNESSES

In the iStory system, the content distribution and synchronization is at the level of media objects. Media objects are distributed to interactors, and synchronized with a TimelineController. Centralized synchronization requires a stable and fast network infrastructure to ensure that time-based 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 iStory system is running on a poor network.

An implicit assumption has been made in the design and implementation of the iStory 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 iStory framework for an interactive program which does not require any visual presentation, e.g. an interactive radio program.

8.4 FUTURE DEVELOPMENTS

StoryML doesn't associate any user input to a specific media object, but to an interactor instead. The interaction specification stays on the level of StoryML

document. Considering that a media object in StoryML can be a MPEG-4 stream, the media object itself will contain another layer with interactive objects with which the user can interact. How to deal with such multilayered interactive content is not answered in this project but has to be addressed in future development.

Another untouched issue is multi-user interaction. In a media presentation environment, especially the home environment, different user profiles should be considered in the future development: children, parents, guests, neighborhoods, etc. Different users would like to have different favorite interface devices or interface "look&feel". Multiple users may enjoy the content together: cooperatively or competitively in the home or at different places. This introduces an interesting direction: The environment, and the distributed interfaces in the environment, should be adaptive to different users.

For presenting interactive content in a distributed environment, another possible alternative approach is so-called scalable interactors. Instead of dispatching media objects to the interactors, the interactive content document itself, e.g., a StoryML document will be directly distributed 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 iStory system has. It is very interesting, though it is not clear yet how these same documents rendered by different interactors can keep updated simultaneously when the user interacts with one of them.

...

It is 6:30pm already.

TOONS is over, but it promises that more episodes are coming. "Tony, when will be the next TOONS show?" Xiaoxiao enjoyed the program very much.

"Next week, same time." Tony says.

REFERENCES

[1] Allen, J.F. Maintaining Knowledge about temporal intervals. Commun. ACM, Vol. 26, pp.832-843, Nov. 1983.

[2] Blair, G.S.; Coulson, G.; Papathomas, M.; Robin, P.; Stefani, J.B.; Horn, F.; Hazard, L. A Programming Model and System Infrastructure for Real-Time Synchronization in Distributed Multimedia Systems. *IEEE Journal on Selected Areas in Communications*, Jan., 1996. Vol 14, Issue 1, pp. 249-263.

[3] Blakowski, G.; Steinmetz, R. A Media Synchronization Survey: Reference Model, Specification and Case Studies. *IEEE Journal on Selected Areas in Communications*, Jan. 1996. Vol. 14, Issue 1, pp 5-35.

[4] Bosak, Jon. *XML, Java, and the Future of the Web*. Available on the Web: < http://www.ibiblio.org/pub/sun-info/standards/xml/why/xmlapps.htm>

[5] Buford, J.F.K. Architectures and Issues for Distributed Multimedia Systems. In: Buford J.F.K. *Multimedia Systems.* New York, New York, USA: ACM Press, 1994, pp.45-46.

[6] Bukowska, M. *Winky Dink Half a Century Later. Interaction with Broadcast Content: Concept Development Based on an Interactive Storytelling Application for Children*. Final report of the post-masters program: User-System Interaction. Aug. 2001. ISBN 90-444-0116-5.

[7] 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.

[8] Choi, S.; Chung K.; Shin, Y. Distributed Multimedia Presentation System Based on the MHEG. *Proceedings, 12th International Conference on Information Networking, 1998*. Jan. 21-23, 1998. pp.403-406.

[9] Coomans, M.K.D.; Timmermans, H.J.P. Towards a Taxonomy of Virtual Reality User Interfaces. *Proceedings, IEEE conference on Information Visualization (IV '97).* 1997. London.

[10] Coutaz, J. PAC-ing the Architecture of Your User Interface. *DSV-IS '97, 4th Eurographics Workshop on Design, Specification and Verification of Interactive Systems*, Springer Verlag Publ., pp. 15-32.

[11] Duke, D.J.; Herman, I. A Standard for Multimedia Middleware. *Proceedings of the 6th ACM International Conference on Multimedia '98*. Bristol, UK. pp 381-390.

[12] Eric S. Interactive Toy characters as Interfaces for Children. *Information Appliances and Beyond: Interactive design for consumer products*. Morgan Kaufmann Publishers, 2000.

[13] Evain,J,P, *The Multimedia Home Platform - an overview*. EBU Technical Review, Spring 1998.

[14] Future TV project. < http://www.tml.hut.fi/Research/future-tv/index.html>

[15] Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, Massachusetts: Addison-Wesley, 1994.

[16] Herrmann, L. *Immersive Broadcast: Concept and Implementation.* Philips LEP Techinical Report C 2000 748.

[17] IBM XML Suite of Java Beans <http://www.alphaworks.ibm.com/alphabeans>

[18] ISO (International Organization for Standardization). ISO 8879:1986(E). *Information processing - Text and Office Systems - Standard Generalized Markup Language (SGML). First edition -- 1986-10-15. [Geneva]:* International Organization for Standardization, 1986.

[19] IST program. NexTV Project. <http://www.extra.research.philips.com/euprojects/nextv/>

[20] James H., Edwin h., David K. Distributed Cognition: Toward a New Foundation for Human-Computer Interaction Research. *Transactions on Computer-Human Interaction*, ACM Press, New York, vol.7, No. 2, June 2000, pp. 174-196.

[21] Jeremy S. G. *A taxonomy of interactivity*. Available on the Web: <http://www.itvnews.com/research/creation.htm>

[22] Koenen R. MPEG-4, Multimedia for Our Time. *IEEE Spectrum*, Vol. 36, No. 2, February 1999, pp. 26-33.

[23] Koninklijke Philips Electronics N.V. *Special item: Ambient Intelligence.* Available on the Web: <http://www.research.philips.com/generalinfo/special/ambintel/>

[24] LEGO Mindstorms Robotics Invention System (RIS) <http://mindstorms.lego.com/>

[25] Little, T.D.C. Time-Based Media Representation and Delivery. In: Buford J.F.K. *Multimedia Systems*. New York, New York, USA: ACM Press, 1994. pp. 175-200.

[26] Luetteke,G. *The DVB Multimedia Home Platform*. Philips Consumer Electronics, Hamburg, November 1998.

[27] Mallart, R. *Immersive Broadcast Reference Application*. White Paper, Sept. 1999.

[28] Meyer, T.; Effelsberg, W. Steinmetz, R. A Taxonomy on Multimedia Synchronization. *Proceedings of the 4th Workshop on Future Trends of distributed Computing Systems, 1993.* Sept. 22-24, 1993. pp. 97-103.

[29] MPEG. *Overview of the MPEG-4 Standard.* Available on the Web: <http://www.cselt.it/mpeg/standards/mpeg-4/mpeg-4.htm>

[30] NexTV. *Deliverable D1 WP3: Application Version 1.* March 2001.

[31] NexTV. *Deliverable D2 WP1: Requirements for the Selected Application and its User Interface*. June 2000.

[32] Norman D.A.; Draper S.W., User Centered System Design. *New Perspectives on Human Computer Interaction*, Lawrence Erlbaum Associates, Inc., Hillsdale, NJ, 1986.

[33] Pfaff G.E. et al.: User Interface Management Systems, Pfaff, G.E. ed., *Eurographics Seminars*, Springer Verlag, 1985.

[34] 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.

[35] SGI. *SGI Mediabase: Intelligent Media Streaming for Intranets and the Internet*. Available on the Web: <http://www.csupomona.edu/~glen/mediabase/sgi.pdf>

[36] Solorzano, J. *leJOS: Java based OS for Lego RCX*. Available on the Web: <http://lejos.sourceforge.net/>

[37] Sun Microsystems, Inc. *Java 2 SDK, Standard Edition Documentation Version 1.3.1.* Available on the Web: <http://java.sun.com/j2se/1.3/docs/>

[38] Sun Microsystems, Inc. *Java Communications API Users Guide,* Available on the Web: <http://java.sun.com/products/javacomm/javadocs/API_users_guide.html>

[39] Sun Microsystems, Inc. *Java Media Framework API Guide*. Available on the Web: < http://java.sun.com/products/java-media/jmf/2.1.1/guide/ >

[40] Sun Microsystems, Inc. Java Foundation Classes (JFC). <http://java.sun.com/products/jfc/ >

[41] Sun Microsystems, Inc. *Programmer's Guide to the Java 2D API: Enhanced Graphics and Imaging for Java.* Available on the Web: <http://java.sun.com/j2se/1.3/docs/guide/2d/spec/j2d-title.fm.html >

[42] Sun Microsystems, Inc. *The Java Language Specification*. Available on the Web: <http://java.sun.com/docs/books/jls/index.html>

[43] Sun Microsystems, Inc. *The Java Virtual Machine Specification*. Available on the Web: <http://java.sun.com/docs/books/vmspec/index.html>

[44] Sun Microsystems, Inc. *XML technology support in the Java platform – FAQ*. Available on the Web: <http://java.sun.com/pr/1999/03/pr990309-faq.html>

[45] The UIMS Tool Developers Workshop. A Metamodel for the Runtime Architecture of an Interactive System. *SIGCHI Bulletin*, 24(1):32--37, January 1992.

[46] W3C. *Extensible Markup Language (XML) 1.0 (Second Edition).* Available on the Web: < http://www.w3.org/TR/2000/REC-xml-20001006>

[47] W3C. *HTML 4.01 Specification.* Available on the Web: <http://www.w3.org/TR/html4/ >

[48] W3C. *Synchronized Multimedia Integration Language (SMIL) 1.0 Specification*. Available on the Web: <http://www.w3.org/TR/REC-smil/ >

[49] W3C. *Synchronized Multimedia Integration Language (SMIL 2.0) Specification*. Available on the Web: <http://www.w3.org/TR/smil20/cover.html>

[50] Wahl, T.; Rothermel, K. Representing Time in Multimedia Systems. *Proceedings of the International conference on Multimedia Computing and Systems*, 1994, May 15-19, 1994. pp. 538-543.

APPENDIX A. DTD (DATA TYPE DEFINITION) OF STORYML

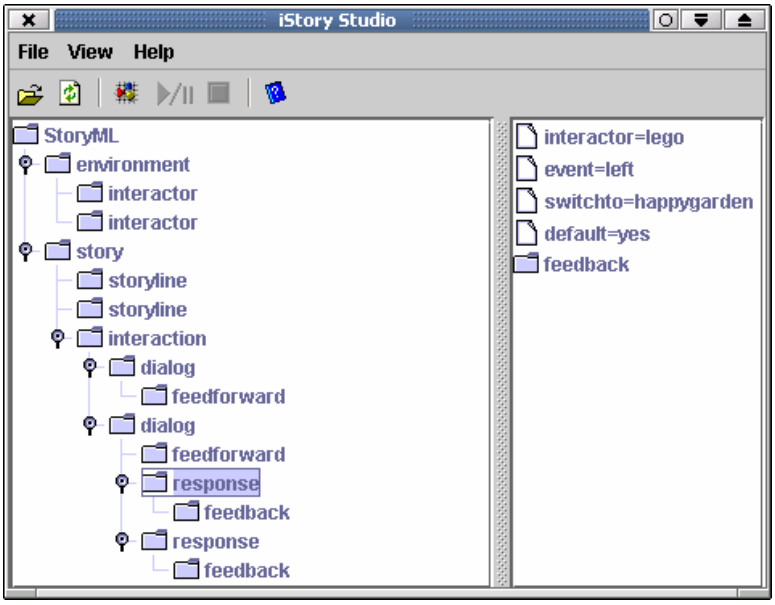
```
<?xml encoding="iso-8859-1"?>
<!–
    StoryML.dtd

    This is the XML document type definition (DTD) for StoryML 1.0.
    Date:   2000/05/30
    Author: Hu, Jun <mail@mrhujun.com>
-->

<!-- Generally useful entities -->
<!ENTITY % id-attr "id ID #IMPLIED">
<!ENTITY % title-attr "title CDATA #IMPLIED">
<!ENTITY % media-attr "
    src         CDATA       #IMPLIED
    content     CDATA       #IMPLIED
    interactor  IDREF       #IMPLIED
    type   (audio|video|audiovisual|text|image|graphics|behavior)
           'audiovisual'
">
<!--=================== StoryML Document ==========================-->
<!--
     The root element StoryML contains all other elements.
-->
<!ELEMENT StoryML (environment?,story?)>
<!ATTLIST StoryML
    %id-attr;
>

<!--=================== Environment Element =======================-->
<!ELEMENT environment (interactor*)>
<!ATTLIST environment
    %id-attr;
>

<!--=================== Interactor Element ========================-->
<!ELEMENT interactor EMPTY>
<!ATTLIST interactor
    %id-attr;
    type    (audiovisual|robot)    "audiovisual"
>

<!--=================== Story Element =============================-->
<!ELEMENT story (storyline*, interaction+)>
<!ATTLIST story
    %id-attr;
    %title-attr;
>

<!--=================== Storyline Element =========================-->
<!ELEMENT storyline EMPTY>
<!ATTLIST storyline
    %id-attr;
    %media-attr;
>

<!--=================== Interaction Element =======================-->
```

```
<!ELEMENT interaction (dialog*)>
<!ATTLIST interaction
    %id-attr;
>

<!--==================== Dialog Element ==============================-->
<!ELEMENT dialog (feedforward*, response*)>
<!ATTLIST dialog
    %id-attr;
    begin    CDATA   #IMPLIED
    end      CDATA   #IMPLIED
    wait     CDATA   #IMPLIED
    type     (immediate|delayed) "delayed"
>

<!--==================== Feedforward Element =========================-->
<!ELEMENT feedforward EMPTY>
<!ATTLIST feedforward
    %id-attr;
    %media-attr;
>

<!--==================== Response Element ============================-->
<!ELEMENT response (feedback*)>
<!ATTLIST response
    %id-attr;
    interactor      IDREF       #IMPLIED
    event           CDATA       #IMPLIED
    storyline       IDREF       #IMPLIED
    action          (switchto|change)    "switchto"
    changecontent   CDATA       #IMPLIED
    default         (yes|no)    "no"
>

<!--==================== Feedback Element ============================-->
<!ELEMENT feedback EMPTY>
<!ATTLIST feedback
    %id-attr;
    %media-attr;
>
```

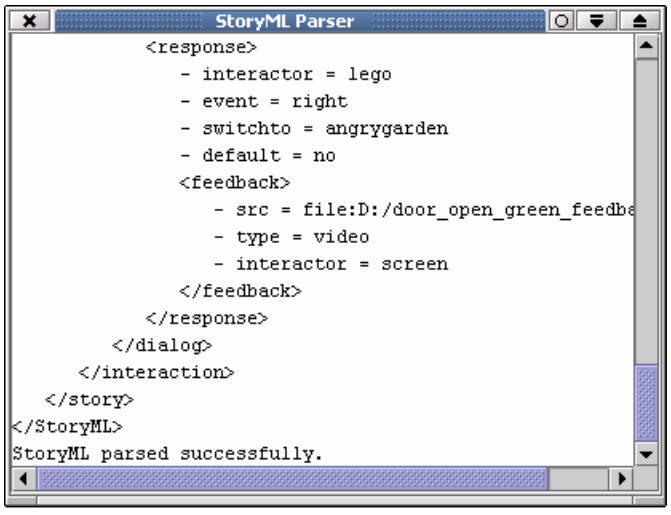
APPENDIX B. TOONS IN STORYML

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

<!--=======================================
    This is the XML document for TOONS.
    Date:   2000/05/30
    Author: Hu, Jun <mail@mrhujun.com>
========================================-->

<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>
```

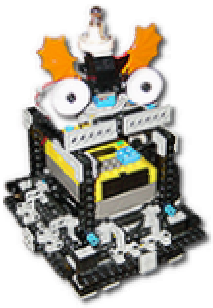
APPENDIX C. SCREENSHOTS FROM ISTORY



A StoryML document is first parsed by the IBM XML parser.



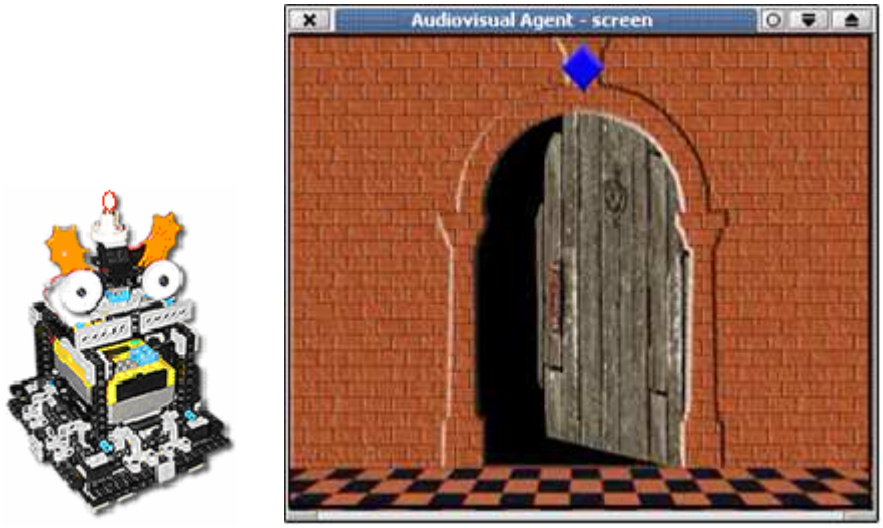
The StoryML parser creates the internal objects according to the DOM (Document Object Model). After pre-fetching certain amount of content, the story is now ready to show.



TOONS starts. It seems that Tony is not so interested –
Tony has a light on his head but now it is off. He is sleeping.

Blast! Tony is woken up by the sounds and his head light is
now dimly on.

The girl is standing in front of two doors. “Which door should I enter? They all look the same...The left? The right? Hmm, I am not sure...Tony, maybe you can help me...” The story catches Tony’s attention. Tony was moving back and forth, flashing his head light, now he is standing there with his head light brightly on.



The user can make decision by tapping Tony’s left or right touch sensors. The user selected the left door, so the left door is opening for the girl now.

Behind the left door there is a beautiful garden with colorful trees and puffy bushes. Tony is getting tired and his head light becomes dim again.

The software interactor takes the role of Tony if he is not available. The user interaction can be switched between Tony and

the software interactor on the fly according to the current configuration of the environment or the preference of the user.