UNIVERSITY OF OSLO
Department of informatics

A Conceptual Framework for Development Iterations of Digital, Interactive Art;

System Response Patterns and Multi-Narrative Temporal Data based Presentation

Master thesis
60 credits

Martin Havnør

31. January 2006

Abstract

Digital, interactive art (DIA) installations often consist of both live and pre-recorded audiovisual material in their presentation of non-linear narrative experiences. Designing and implementing the software supporting the necessary manipulation of such media is complex. Installations of a broad experimental scope require qualifications in a number of fields besides computer science. This further complicates the process by adding multi-discipline communication and negotiation.

This thesis addresses two problems related to this process; choice, classification and identification of interactivity components, and choice of software platform.

A review of existing literature on interactivity theory is used as a base for defining a theoretical framework supporting part of the development process. The framework consists of a software design pattern based on the Model View Controller, and a set of empirically based “System Response Patterns,” a concept contributed by this thesis.

Specific to DIA installations using temporal presentational data, the framework should help in choosing between different means of communication between audience and installation, and be useful in the development of related software.

Additionally the use of Java in this context is thoroughly explored through the development of a case installation with the Java Media Framework. A methodology based on Agile Software Development is employed.

Acknowledgements

I would like to take this opportunity to express my gratitude to the following that have contributed or helped in realizing this thesis:

Ole Smørdal, my supervisor whose academic and technical insight has been of tremendous assistance every step of the way.

Idunn Sem for valuable input and co-ordination of many Tapet testing processes.
Andrew Morrison for valuable input on all art and media related issues.
Jo Herstad, my internal supervisor.
Jan Arild Dolonen for valuable technical assistance.
Even Westvang for shepherding me into the Tapet project.

My co-workers from the Tapet project (In alphabetical order):
Siri Gjønnum
Per Christian Larsen
Andrew Morrison
Inger Reidun Olsen
Knut Qvale
Idunn Sem
Ole Smørdal

And finally Intermedia at the University of Oslo for providing me with the opportunity to write this thesis and work on the Tapet project.

Table of contents

List of figures

Preface

My fascination with computer generated or manipulated graphics started as early as 1990, with the programming of simple ASCII-based animations on a C64. It developed into demo and intro-coding, a discipline of programming requiring extreme attention to detail and performance.

It was through conversing on this common background from the Norwegian demo-scene with Even Westvang that I first encountered interactive art. We met an early Monday morning, or more precisely a very late Sunday night.

He was working in the studio of the Intermedia centre, on a project called Karikuri. The core of the project was a large canvas. A Mac was projecting images of basic geometric figures onto the canvas, which in turn was taped by a video camera. By moving in front of the canvas, you could interact with the geometric figures!

I was at the time searching for a subject for my master’s thesis, and intrigued I asked him if he knew of any possibilities in related projects. He did, and shortly after I was both involved with the Tapet project, had found an extremely interesting subject for my thesis and an excellent supervisor.

1 Introduction

As computers have become common in homes and workplaces, the digital dimension has entered into virtually every thinkable domain. Art is no exception. Museums are adding digital works to their collections. Exhibitions consisting exclusively of digital art are also becoming more common.

Digital, interactive art

An emerging trend in the digital art world is interactivity; letting the audience themselves take part in the process of shaping art. By using properties of the audience or their actions as integral parts of their artworks, artists are offering a potentially unique experience to every visitor. Additionally, the traditional role of the audience as a passive observer in terms of actions may be challenged, forcing the audience to participate at some level.

Information Systems Design (ISD)

Works of interactive digital art are often called ‘installations,’ referring to the fact that they may require entire rooms of physical space as opposed to sculptures, paintings and the likes. Situated next to the physical space open to the audience, are usually computer systems receiving input from sensors, display images, play sounds and coordinate the different components of the installation. These computer systems can grow to considerable levels of complexity, making the process of designing and implementing them rely on a predictable system of program code and concepts. In this thesis I want to emphasize this process, and relate ISD to the process of art creation.

The Java development platform

The applications running interactive art installations can be implemented using a range of different software; among them is the free Java platform by Sun Microsystems. Java was first introduced as a platform independent programming language, aimed at designing and implementing small programs that could run in web browsers; applets. This happened at a major conference in 1995 and generated enormous interest due to the hype surrounding the World Wide Web at the time. Java was built on earlier programming languages such as C++ and Simula in the sense that it was object oriented.

From there Java has evolved into one of the most versatile programming platforms available, and among the most common languages taught at computer science faculties world wide. In addition to the regular application class software, Java code can be compiled into ‘applets’ that run in web browsers, ‘midlets’ that run on mobile phones, ‘servlets’ that are server side components of web pages and more. In this thesis I focus on Java as an implementation platform for the software running interactive art installations.

Objectives

The artist behind an installation of digital interactive art while having conceived the idea will often lack the technical expertise to perform all of the subtasks involved in the actual creation of the artwork.

Video-technicians, sound-specialists and others are involved at different stages in the production. One of the problems facing the artist is finding a way to properly articulate the interactivity component of the installation, in a way comprehensible to everyone involved.

The structuring of the interactivity-component of the installation would also help its incorporation into the ISD process. This thesis aims to create a conceptual framework based on interactivity theory and ISD to help this structuring. More specifically, this is accomplished by applying Lee's (2000) model of interactivity to the setting of digital, interactive art.

Through implementing the case installation in Java, the ramifications of this choice of development platform – both to process and result is mapped.

Research method

A research method resembling the learning cycle presented by Smørdal and Kaasbøll (1996) was utilized in the development process of the case installation. The learning cycle involves a circular pattern beginning with a design process, followed by implementation built on the design. When implementation is done, testing and observing the results are performed and finally analyzed. The results of the analyses are then used in another iteration of design, fulfilling the circle. (Smørdal and Kaasbøll 1996)

The initial input to the design process was threefold; field study at the 25th Ars Electronica[1] festival, an extensive literature review of core concepts, and finally software design process theory acquired partly through literature review, and partly through own experience. The implementation process consisted of an actual interactive art project called "Tapet." Tapet was then tested with several test subjects, their experience videotaped. Their feedback, combined with analyses of the video led back to the design table and so on.

[1] Ars Electronica is an annual festival of electronic art held in Linz, Austria

Figure 1 The Tapet installation set up in the Intermedia studio.

The end result of Tapet was a physical installation as big as a medium-sized living room, consisting of a massive wallpaper, a display canvas, two cameras and numerous computers out of view running the actual installation. Java was chosen as development platform, and in fact used for all the different pieces of the final system of applications.

Research findings

Through the development process, several interesting findings were made. The software was divided into three main parts according to the Model View Controller pattern, and each of these components was proven possible to implement using Java. One of the contributions of this thesis is a modified version of this pattern, better suited to the temporal data types of installations similar to Tapet.

During the literature review of interactivity, the definition of Jae-Shin Lee (2000) was found to be the most suitable to digital interactive art. However, some mismatch was found, and the thesis constitutes a new definition aimed specifically at artworks similar to Tapet. Finally the concept of System Response Patterns and five examples as used in the Tapet installation were identified, while researching the conceptual framework surrounding interactivity in the setting of digital art.

Thesis outline

This thesis is divided into three main parts.

The first part comprised of chapter two and three is built on the literature review. Background information and the necessary definitions are covered, as well as a look at the existing technologies in the realm of digital, interactive art that are of relevance to this thesis.

The second part, comprised of chapter four, contains a description of the research problems encountered and the method employed to solve them.

The third and final part spans chapters five through seven. It contains a description of the development process and result of Tapet, a discussion over the research problems and finally conclusions along with a list of contributions.

2 Definitions and background information

In this chapter the main conceptual terms of importance in this thesis will be defined. Existing literature and definitions are evaluated and chosen with a view to selecting that which best fit the case project Tapet, as well as digital, interactive art in general.
The terms that are visited upon are interactivity, digital art, interactive digital art, Model View Controller, embodied Interaction, narrative structures, temporal data and state machines.

2.1 Interactivity

Through a review of existing literature and theory on interactivity, Lee's (2000) model is chosen and applied to the setting of digital, interactive art to map the effects the different forms of interactivity applied has on an audience. This section provides a description of existing definitions and models of interactivity, and how they lead to Lee's model.

Radio, newspapers, computer programs and games are terms that we all know well in contemporary society. The actual origin of the word "interactivity" however, is not so easily determined.

Commonly the word is thought to stem directly from 'interaction', meaning action involving two or more parts, in Norwegian 'vekselvirkning, samspill, gjensidig påvirkning' (Jensen 1998)

According to the Curator at the National Gallery of Iceland, Margrét Elísabet Ólafsdóttir (2001), the actual word was first used around 1960, in reference to the fact that scientists had been able to interrupt the operations of a computer.

> "*They called the interruption an interactivity and decided (I hope you'll forgive me the simplification) to focus on the partnership of man and machine in further development of the computer.*" (p 1)

The term interactivity is of late highly associated with the definition given to the word by the computer science community;

> "*In computers, interactivity is the sensory dialog that occurs between a human being (or possibly another live creature) and a computer program.*" (Wigmore 2005) (p. 1)

Prior to the 80's and 90's however, before the word became mainstream and widely used, other definitions were just as, if not more accepted. This section starts by reviewing many previous descriptions, definitions and dimensions of interactivity, followed by an

examination of how to observe interactivity empirically – whether directly or comparatively.

Existing definitions and dimensions of interactivity

Researchers of several fields of knowledge have contributed to the exploring and mapping interactivity, its definitions, dimensions, uses and effects. Of these, the fields of communication, business school and advertising have been major contributors. Additionally, HCI (Human Computer Interface) and computer science have had a major impact; particularly the last 10-15 years. (Lee 2000) This section looks closer at the contributions of these fields to the conceptualization of interactivity.

Sociology

In sociology, interaction is strictly described as an exchange of actions between two individuals, who are both aware of themselves and find it reasonable that the other individual is as well. (Jensen 1998)

Although sociology studies have not directly approached the concept of interactivity, we can acquire useful insights from the sociological meaning of 'interaction'. In the International encyclopedia of communications, Duncan (1989) gives a definition of interaction from a sociology point of view;

> "*Interaction makes up a basic constitute of society.*" (p. 326)

Jensen (1998) notes that

> "*The basic model that the sociological interaction concept stems from is thus the relationship between two or more people who, in a given situation, mutually adapt their behavior and actions to each other.*" (p. 188)

Media and communication

Related but notably different, is the definition used by the media and communication societies. While the sociologists focus on both parts of the exchange being aware and sentient, in essence human, interaction is here the process undergone by an audience or a set of viewers, towards the contents of media they are observing. This regardless of whether any new (as defined in computer science) 'interactive media' are present. (Jensen 1998)

Kiousis (1999) saw interactivity as an attribute of the channel or medium which enables a dynamic interdependence between senders and receivers in communication. As early as 1948, Wiener (1948) pointed out the important role of feedback in communication studies. This concept is brought up again by Rafaeli (1988) who introduces the importance of responsiveness, based on Wiener's feedback. He suggests that in a set of information exchanges, the interactivity is an expression which states to what degree

messages after the second, refers to any previous messages. (p. 111) Another communication scholar who saw feedback as a prime aspect of interactivity was Heeter (1989), who set up a matrix of six dimensions of interactivity; feedback being one of them. Williams et al. (1988) argues that interactivity is a unit that can be used in describing the amount of control participants in a communicative process have of their respective roles in the process. (p. 10)

These studies of interaction based in media and communication sciences focus on message relationships and emulating FTF (Face to face) communication; thus not entirely relevant to modern interactivity studies where the technological characteristics of the medium are important factors. However they did contribute to the future development of conceptual models of interactivity by examining content of interactive media, and trying to link it to psychological and behavioral attributes of users.

Laurel (1993) is among the earlier researchers exploring interactivity from a media perspective. She argues that the experience and structures of acting in a theater can be used as a base for designing human-computer communication and interaction. She sees interactivity as an actual experience that can be aimed for and achieved by this. Similarly Laurel also defines three characteristics of interactivity; the frequency of opportunities for interaction, the number of available choices and the significance attributed to each of these choices.

DeVries and Wheeler (1996) argues that in distance education where FTF communication is not possible, the engagement put forth by the pupils is the core of interactivity. While not directly applicable to other settings, one could also argue that this is the case in any learning situation.

Walther and Burgoon (1992) recognizes that in this line of research (media), FTF is considered the ‘best’ form of interactivity. When evaluating mediated communication, researches measure the similarity between the mediated communication in question, and FTF. (pp. 50-88)

Bordewijk and Kaam (1986) proposed a four-party typology of information models that has later been applied to defining interactivity. Among those who have, McMillan (2002) describes the typology when attempting to offer a new two-dimensional definition of cyber-interactivity;

> “*Many students of communication have been introduced to the Bordewijk and Kaam (1986) four-part typology of information traffic through McQuail’s (1994) mass communication theory textbook. The key element of the Bordewijk and van Kaam typology is control. One dimension of the model is defined by control of information source and the other by control of time and choice of subject. For both of these variables, the Dutch scholars suggested that control may reside either in a central source or with the individual.*”

McQuail (1994) builds on the typology of Bordewijk and van Kaam's, further developing the model. McQuail suggests that based on whether the two types of control reside in a central source or with the individual, 'allocution (transmission)', 'registration', 'consultation' and 'conversation' could be defined. Among these, 'allocution (transmission)' is closely related to mass media and 'conversation' is more like interpersonal communication.

Applying the concept developed by McQuail, Jensen (1998) defines four types of interactivity in computer-mediated environment.
He defines interactivity as

> "*A measure of a media's potential ability to let the user exert an influence on the content and/or form of the mediated communication.*" (p. 201).

According to Jensen, a communication pattern of the 'transmission' type can be found in TV and real time radio, and the 'conversation' type in email and Internet Relay Chat (IRC). Jensen then uses these four types of interactivity in explaining the definitions of interactivity suggested by others. For instance, Jensen argues that Rafaeli's (1988) responsiveness primarily refer to 'registration' communication; centrally controlled information provided by the consumer.

Ha and James (1998) recognize 'exchange' and 'mutuality' as previously considered key elements of interactivity when applied to a CMC (Computer Mediated Communication) context. They argue that in such a setting these are invalid, and thus propose interactivity as

> "*The extent to which the communicator and the audience respond to, or are willing to facilitate, each other's communication needs.*" (p. 8)

They also suggest 'playfulness', 'choice', 'connectedness', 'information collection' and 'reciprocal communication' as the five dimensions of interactivity in CMC.

Business school

Haeckel (1998) presents several definitions of interactivity from attendees at the May 1996 Harvard Business School Conference on Marketing Interactivity. One example is:

> "*A two-way dynamic dialogue, person-to-person communication permitting feedback and exchange between two entities that changes the state of at least one of them.*"

Day (1998) argues that

> "*the essence of interactive marketing is the use of information from the customer rather than about the customer*" (p. 47)

pointing out the importance of mutual, two-way communication. From a marketing point of view, the focus of interaction lies in establishing communication with potential customers to create compelling consumer experiences.

The importance of managers' use of electronic messaging for ongoing management groups performing a cooperative task is identified by Zack (1993). He presents four dimensions of interactivity; 'simultaneous and continuous exchange of information', 'use of multiple, non-verbal cues', 'potentially spontaneous, unpredictable and emergent progression of remarks', and 'ability to interrupt or preempt'. When evaluating the interactivity of electronic messaging, he sets face-to-face communication as a ‘consummation of interactivity’.

Computer Science

Another approach in defining interactivity is focusing on the different technological features of the available media.

In computer science, nearly all programs or applications developed have some form of interactivity embedded. Kearsley and Halley (1986) mentions exceptions such as drivers and certain sales demos, but conclude that any program that requires used input is by definition an interactive program. (p. 13)

Human Computer Interaction

Human-computer relationship and interaction is central in human-computer interaction (HCI) studies.

Jens F. Jensen is a major figure in HCI research. He reports that the “style of control that exists between the human and the computer” has been the key determinant of interactivity in the field of human-computer interfaces. (1998)

In the studies of virtual reality, Sheridan (1992) notes similar conceptions. When identifying five variables that help induce the feeling of telepresence (a feeling of being present in an environment generated by mediated means), he includes three technological factors; the extent of sensory information, control of sensors relative to environment, and the ability to modify the physical environment. The other two are task-, or context-based: task difficulty, and degree of automation.

A definition of interactivity is put forth by Steuer (1992);

> "*The degree to which users of a medium can influence the form or content of the mediated environment.*" (p. 80)

He identifies 'interactivity' and 'vividness' as determinants of telepresence. Steuer's three dimension of interactivity are 'speed', 'range' and 'mapping', which are all technological elements.

Goertz (1995) cited in Jensen (1998) identifies four dimensions of interactivity that have much in common with those suggested by Laurel (1993). The four dimensions are the degree of choices available, the degree of modifiability, the quantitative number of the selections and modifications available and the degree of linearity.

Bezjian-Avery et al. (1998) continue the research into the idea of interactive systems having a degree of linearity. They argue that the ability to control information is the fundament of the interactivity concept.[2] They conclude that information flow in traditional advertising is linear, while that of an interactive system is not; it can be networked, circular or similar.

With the Internet and World Wide Web becoming more widespread in use, interactivity is becoming more connected to communicating with the backbone database of a web site. In line with this new tradition, Klein (2000) views bandwidth as a key element of acquiring interactivity.

Hutheesing (1993) notes

> "*Professionals often regard interactive media as mechanisms for delivering image, text and sound data in which the user interacts with the database."* (p. 244)

Among the many definitions and dimensions of interactivity that exist, Lee (2000) found they could be divided into two main groups, something that largely had been ignored till then. The running of experiments as well as forming explanations and definitions across these two main types has caused much of the confusion surrounding interactivity, and has also according to Lee delayed the exploration into more specific and detailed descriptions of interactivity. Lee defines these two classes of interaction as follows:

- User-user communication
- User-medium communication

User-user communication denotes interaction between at least two human beings, without setting restrictions on the medium they use for communication. This could be as simple as a phone call, a bulletin board on the internet or internet relay chat.

User-medium communication is interaction between a user and a medium such as a machine, computer of some sort. From the computer science perspective this form of interaction is the core of Human Computer Interaction. (Lee 2000)

Consequently, Lee suggests a definition that encompasses the two forms;

---

[2] Bezjian-Avery, A., B. Calder, et al. (1998). "New media interactive advertising vs. traditional advertising." Journal of Advertising Research **38**(4): 23-32.

> "Interactivity is defined as something that changes according to the user's input."

While fairly simple, the definition does not restrict the other side of the communicative process; user-user or user-medium. Dimensions such as change and feedback are implied, while technological aspects such as Klein's bandwidth or synchronicity are not. (Lee 2000) The central idea of the definition, according to Lee, is that to be interactive changes should be made in accordance with the user's intension.

Observing interaction

Beyond that of defining interactivity, some way of empirically studying or a way of comparing different forms of interaction is needed for the idea to be useful in a conceptual framework.

Interactivity as a concept has traditionally been, and still is, difficult to observe or conceptualize. There are no apparent quantifiable characteristics that stand out as universal identifiers, and the vast number of definitions available furthers these difficulties.

Among the definitions that have been suggested, many have provided dimensions that further explain the contents of the term interactivity. Lee (2000) identifies failure to organize these dimensions as part of the reason why there has been so little progress in operationalizing interactivity.

Jensen (1998) also recognizes the proposed dimensions as a major problem when trying to deal with measuring interactivity on a practical basis. He argues that Heeter's (1989) six dimensions are overlapping to such a high degree that it becomes practically impossible to use them in actual studies.

Lee (2000) goes through several studies that attempt to perform such measurements, their results and shortcomings. These studies can be divided into two categories, based on whether they investigate interpersonal communication or user-medium communication. Lee argues that much of the shortcoming of these studies lies in their failure to cover both the user-user and user-medium aspects of interactivity in their subjects.

Rafaeli ((1988), (1993), (1997)) was among the first to conduct such studies. Based on his concept 'responsiveness', he examined the relationship among message threads. Considering the forum threads as pure messengers and not a part of the actual interaction, he focused only on the interpersonal communication performed by the users of the forum through each message thread.

Schultz (1999) conducted a content analysis of 100 U.S. online newspapers. He believed that interaction could only occur between people, that machines can only mediate the communication. For this reason, he only examined the availability of options that facilitate interpersonal communication. He chose the existence of interpersonal

communication modules on the websites such email, chat rooms, online polls and surveys as the only subcomponents of interactivity.

Similarly, Ha and James (1998) conducted a content analysis of 110 business web sites. They prepared five dimensions of interactivity based on interpersonal communication options in much the same way as Schultz, also ignoring the user-medium interactivity offered by the sites.

On the other hand, Shaw et al. (1993) performed an experiment attempting to quantify the effect of computer mediated interactivity on idea generation, exclusively based on interaction with computers. They did not actually measure the level of interactivity; rather it was assumed that providing navigational buttons and relevant information would increase it. Three programs with differing levels of communicative options were made, and offered to three groups of test participants. They expected the higher level of interactivity-generating options to infuse a better set of ideas; however they were unable to identify such an improvement.

According to Lee (2000), failing to realize that interactivity varies among people is a 'crucial shortcoming' of these studies. The scholars working the experiments considered the level of interactivity to be fixed by the availability, or lack thereof, of certain technical functionalities

Newhagen et al. (1995) argues that increasing the number of communication options on a website does not equal increasing its level of interactivity. As an example, they mention that the editors of Newscast do not even look at the emails they receive from their audience, although they encourage this kind of contribution in the form of sending them email.

Lee identifies a threshold where the number of communication options will overload, causing the experience of interactivity to diminish rather than increase due to overwhelmed users, or staff not being able to cope with the amount of incoming messages. Additionally, he argues that the use of communication abilities of web sites is not exclusively linked to what the users are able to do, but what they want to do. Participation in online discussion groups requires time and effort, and to those not willing to invest that time and effort, the availability of such a feature does not change the interactivity of the site.

Reviewing many of the previous studies that attempted to measure interactivity, Lee (2000) argues that the line of thinking started by McMillan[3] is the key to a proper analysis and measurement of interactivity. She suggests that rather than focusing on the existence or quality of communicative components on a website, one should focus on the user perception of interactivity. She performed her own analysis of selected websites, and had test participants document how they perceived the interactivity of the same websites.

[3] This paper was first presented at the annual conference of the International Communication Association (ICA) in 1999.

Finally comparing the two, gave a better idea of the interactivity-level offered. (McMillan 2002)

Having stated that perceived interactivity is a main characteristic of interactivity, Lee goes on to describe the communication context/setting as a contributor to mentioned perception. As an example, people of varying levels of skill and experience with the use of electronic mail will rate the interactivity of e-mail differently. The higher the level of proficiency with a specific communication technology, the higher its interactivity was rated by the user. (Fulk, Schmitz et al. 1990)

These two form the core of Lee's model of interactivity; *setting* and *user perception*.

2.2 Digital art

Digital art is a central concept in this thesis, and is thus entitled to a proper explanation. However, the term has not been the subject of much discussion or attempts at definitions. The scope of art touched upon by this thesis is limited to interactive installations, and in some aspects interactive installations using embodied interaction as a means of input. To prevent unnecessary further limitation, I wanted to adopt a definition of digital art that was as broad as possible.

UNESCO (2004) wanted to expand their existing definition of digital art;

> "*While this (digital art) covers a wide range of production from interactive installations to websites as artworks and digital video productions, it does not include the scanning or documenting, cataloguing and marketing of traditional art forms. The digital technology needs to be a part of the artwork, without which the work would not exist.*" (p. 1)

In order to include more of both potential art and artists, UNESCO wanted to expose more of the African population to digital art and the possibilities it offers. As a tool to help achieve this, they wanted to expand their definition to art that is digital either in production, or presentation.

According to Wikipedia (2005) digital art is art that has gone through one or more non-trivial computing processes. This includes computer-generated imagery such as fractals, but usually not digitized media in itself; only if part of a larger system.

Wikipedia mentions the broader definition offered by the Austin Museum of Digital Art (2005):

> "*AMODA defines digital art as art that uses digital technology in any of three ways: as the product, as the process, or as the subject.*"

Their definition is not intended to exclude but to encompass as much as is reasonable:

> "*We seek to expand the public's definition of digital art (and our own) in order to appreciate the truly vast and far-reaching impact of digital technology on art, on the world, and on ourselves.*" (AMODA 2005)

Product

AMODA sees digital art with a digital product as any art whose final form is digital in nature. This includes software viewed on a computer, graphical demonstrations, web sites and other works that are displayed on an actual computer. Other products considered digital are works that use nonstandard hardware, such as electronics and robotics. Additionally, works that are made up of parts generally considered digital, such as computer components or mp3-player headphones are considered of a digital nature.

Process

Art that was created using digital technology in the process of its creation would also be digital art. Obvious examples include computer-generated animation, synthesized music, and computer-designed sculpture. While these works might be presented in traditional media (e.g. film, audio tape and marble), their production was facilitated by the use of digital technology.

Less obvious examples include: a painting designed by visitors to a web page; a play which reenacts an e-mail exchange; or music that samples sounds from an arcade game. These are still works which could not exist without digital technology to aid their production.

Encompassed by process is any type of art where digital technology has altered the production of art. This alteration can be subtle or profound, either by impacting traditional production or allowing novel approaches.

Subject

Finally, art that addresses or discusses digital technology is also digital art. A painting depicting a woman using an ATM machine or a song about chat rooms could both be considered digital art. Digital technology need not be the focus of the piece, or even mentioned explicitly. Works that, through their subject, say something about digital technology and its impact on the world fall under this category.

Technically, only the use of functional digital technology in the product would be necessary to describe the type of installations this thesis refers to. However, the broader definition is appealing not only to increase awareness of this fascinating field, but also quite logical; as a painting depicting a scene from the battle of Normandy could be categorized as World War II art, a bust of Turing could be considered digital in nature due to the motive. Regardless of artist, period or place of origin a painting made by quill preserves the quality of being just that – drawn by quill. Similarly a piece of art implemented as computer generated music retains the quality of being computer generated.

2.3 Interactive Digital Art

The definition of interactivity proposed by Lee (2000) is well suited for most cases. Web-site administration modules are things that ‘change according to user input’, and hopefully in a way the user desires. Having a conversation via e-mail will result in answers that vary depending on the previous messages.

In the setting of interactive digital art the wording works very well;

> “*Something that changes according to user input.*”

Usually an installation of digital interactive art will have the user affecting system behavior in some way, but occasionally more than one user will be involved at once – the interpersonal communication observed and used by the system, making the wide definition allowing for both user-user and user-medium interaction useful. The actual meaning of “user” can at times be fuzzy in interactive art, as it tends to challenge the traditional notion of an audience – and even that of an artist. The interactivity itself will put the audience in a participatory role, to some extent putting them in the role of an artist. The actual artist of a piece has a system set up to respond to the audience thus including the audience in the making process.

Lee’s definition of interactivity puts user intention as a central object;

> “*The central idea of the definition is that to be interactive changes should be made in accordance with user's intension.*”

This concept is further explored by Beaudouin-Lafon (2000). He identifies three attributes of interactivity tools, each describing different aspects of the difference in input and output. The ‘best’ change is one that is identical to the input, happens simultaneously and at the same place as the input occurred.

2.4 Embodied interaction

According to Fagerberg et al. (2003), embodied interaction is communication of meaning between users. This communication occurs either directly, or through the system, they argue. An often referred to[4] and widely accepted definition of embodied interaction comes from Dourish (2001);

> *"(Embodied interaction is) the creation, manipulation and sharing of meaning through engaged interaction with artifacts.*"

In an effort to emphasize its importance, Dourish (2000) concludes that embodied interaction is a foundation for new HCI models. Here, artifacts do not only refer to actual physical objects, but social practices as well. By using this definition of embodied interaction, Dourish is able to combine 'tangible interaction' and 'social computing', Fagerberg et al argues.

These two concepts stem from the school of HCI, tangible being interaction where the interaction is distributed over both physical objects in the real world, and an abstract, computer generated world. (Ishii and Ullmer 1997) Social computing is computing where the digital systems are designed primarily for social practice and the construction of meaning through social interaction. (Bannon 1991)

Holden and Dyar (2002) conducts an experiment using embodied interaction through virtual environment training as a tool in rehabilitating stroke patients. Hoffman (2002) documents the use of sensory-equipped gloves in a virtual environment as a way of curing arachnophobia. The interactive artwork iamascope (Fels 2000) is an immersive environment that invites the user to explore the artwork through the use of movement.

Figure 2 Virtual reality used as a tool in curing arachnophobia

[4] Löwgren (2005) "Inspirational patterns for embodied interaction", Fagerberg et al. (2003) "Designing Gestures for an Affective Input"

Computer games have also been a traditional arena for input mechanisms based on body movements. Early games such as Nintendo's "Duck Hunt" from 1984 use primitive although effective input devices mimicking the interaction with real-life tools, while more recent games such as Konami's "Dance Dance Revolution" allows for greater freedom of movement.

Figure 3 Dance Dance Revolution European cup at The Gathering 2005[5]

As these forms of interaction are developing, the 'natural' human movement is increasingly taken advantage of as a more intuitive form of supplying input. People's abilities to act in physical spaces and our familiarity with the manipulation of physical objects are used to create meaningful ways of using the body in a physical and social world. While traditionally the approach to movement-based interaction has been based on making the actions of the user follow a pattern understood by the computer, current developments and ideas through tangible interfaces and social interaction include novel ideas where the computerized system will understand lived human embodiment.

[5] The Gathering is an annual computer festival held in Hamar, Norway

2.5 Narrativity and narrative structures

This section briefly covers the origins of the phenomenon of narrativity, the main research that has been performed on human perception of narrativity through psychology, and finally its role in computer science.

History of the phenomenon: narrative and narrativity

The phenomenon of narrativity is one that has been a part of philosophy, psychology, literature and other schools of knowledge for more than two millennia. Only as recent as the last fifty years however, has the concept of narrative itself been the object of inquiry and studies. (Ryan 2005)

> "*From Aristotle to Vladimir Propp and from Percy Lubbock to Wayne Booth, the critics and philosophers who are regarded today as the pioneers of narrative theory were not concerned with narrative proper but with particular literary *genres, such as *epic poetry, *drama, the *folktale, the *novel or more generally *fiction, short for 'narrative literary fiction'.*" (p. 2)

Ryan argues that French structuralism; especially through Roland Barthes and Claude Bremond were responsible for removing narrative from literature and fiction, recognizing it as a "semiotic phenomenon that transcends disciplines and media."

Narrative in psychology

According to Young and Saver (1998) narrative is a frame, encapsulating all human experience. Our consciousness is fundamentally constituted by an understanding of self and world in story. Young and Saver argue that thinkers such as Aristotle, Barthes and Bruner all recognized the central role narrative has in human cognition. Young and Saver go further however, claiming that the ability to construct narrative is the very core of human personality.

> "*Brain injured individuals may lose their linguistic or visuospatial competencies and still be recognizably the same persons. Individuals who have lost the ability to construct narrative, however, have lost their selves.*" (Young and Saver 1998) (p. 1)

Roland Barthes and Donald E. Polkinghorne respectively sum up the scope of narrative as follows:

> "*The narratives of the world are without number...the narrative is present at all times, in all places, in all societies; the history of narrative begins with the history of mankind; there does not exist, and never has existed, a people without narratives.*" (Barthes 1996) (p. 14)

> "*The products of our narrative schemes are ubiquitous in our lives: they fill our cultural and social environment. We create narrative descriptions for ourselves and for others about our own past actions, and we develop storied accounts that give sense to the behavior of others. We also use the narrative scheme to inform our decisions by constructing imaginative "what if" scenarios. On the receiving end, we are constantly confronted with stories during our conversations and encounters with the written and visual media. We are told fairy tales as children, and read and discuss stories in school.*" (Polkinghorne 1998) (p. 14)

Narrative in computer science

In computer science, the idea of dynamic narrative structures are often focused on, rather than traditional linear narrative structures; traditional mediums such as books, films and recorded audio are well suited for the purposes of linear story-telling. Games are often accentuated as computer applications that may offer non-linear narratives.

Juul (2001) argues that there is an inherent conflict between narrative and interactivity in computer games. Narration and interactivity cannot co-exist Juul argues, as the interaction given in the present will collide with what has already occurred in the past.

> "*There is no such thing as a continuously interactive story.*" (Juul 2001)

This may be true for single-player games, but cannot be seen as universal. Of games, Multi User Dungeons (MUD) and MUD Object Oriented (MOO) are good examples of narratives continuously evolving through user interaction.

Among many studies of the narrative elements in MUDs and MOOs, Malloy (2000) explored the LambdaMOO community based on the LambdaCode system. Malloy identified three possible literary forms of narrative in the system, all made possible by the multi-user concept MUDs are based on.

Non-linear narrative can also be found in interactive art. Similar, but not identical to that of games:

> "*The difference between an interactive game and an interactive work of art is not just in the subject matter. It's also in the program and interface, which are an important part of the expression of a work. Artists working in this field will continue to be at odds with the models and directions of the multimedia industry.*" (Campbell 2000) (p. 134)

When describing the idea of non-linear narrative in the interactive narrative experiment 'Juvenate', Hutchison (2003) refers to Eco (1989) who views 'open work' as one in which the author has deliberately planned to create the need for the reader to make interpretations of their own. The idea is similar to that of interactive games and art; only these require action as well as interpretation.

2.6 Model View Controller

The “Model View Controller” or simply MVC is now a commonly used design pattern in the world of Graphical User Interfaces.

MVC was originally developed by Professor Trygve Reenskaug at Xerox PARC in 1978. Its purpose at that time was to provide convenient GUI support in the programming language Smalltalk. The pattern is still relevant, and has been adopted and embraced by

> “*the distributed design community to also apply on a large-scale architectural level.*” (Larman 2002) (p. 472)

The Java Swing components, as well as the Microsoft Foundation Classes are both based on the MVC pattern. (Wikipedia 2005)

More recently, the pattern has been used for designing interactive websites. The complexity of large web applications is becoming a problem in terms of design; MVC is among the architectures that are considered as a potential solution to these difficulties. (Wikipedia 2005)

The pattern constitutes a clean separation between the three main parts of the user interface object; the business logic lies within the model, the invoking of this logic is handled by the controller usually in the form of events, and finally the view is responsible for displaying correct information and interface options to the user.

This high degree of separation between the different layers allows several forms of views to be attached to the same controller and model.

Figure 4 shows a graphical high-level representation of the MVC design pattern as shown by www.enode.com (2002).

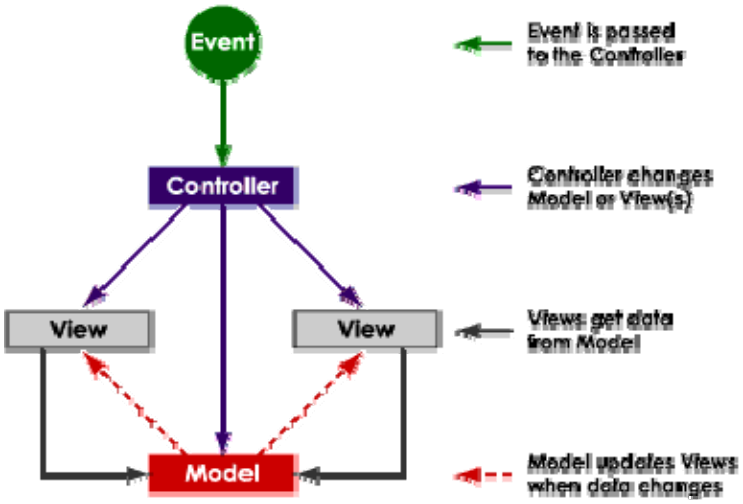

Figure 4 Model View Controller

Cited by most online descriptions of MVC is Dean Helman's text from the Objective Toolkit Pro whitepaper:

> "*The MVC paradigm is a way of breaking an application, or even just a piece of an application's interface, into three parts: the model, the view, and the controller. MVC was originally developed to map the traditional input, processing, output roles into the GUI realm.*"

Traditional: Input --> Processing --> Output
Modern GUI: Controller --> Model --> View

Gathering user input, computation and storage of business logic and visual feedback are separated and handled by model, view and controller objects. The controller receives and interprets user input from various input mechanisms. The input is mapped into commands that are sent to the model and/or view to create the desired change.

The model manages one or more data elements related by a common purpose, responds to queries about its state, and responds to instructions to change state. The view displays both interface options and desired output elements representing change and state in the model.

MVC can also be seen as a framework, made up of several patterns such as Observer and Composite. (Sommerville 2001) A framework, according to Sommerville, is

> "*a generic structure that can be extended to create a more specific sub-system or application.*" (p. 314)

This is indeed the case with the MVC pattern. When designing a system from scratch however, the MVC can be used as a pattern for underlying structures and classes, among other things allowing multiple views attached to the same model. In this setting the MVC is not a pre-fabricated framework supplying call-back methods, but a design principle potent for the world of user interaction; a design pattern.

When describing the benefits of Model-View separation principle, Larman (2002) sees MVC as a pattern of which the aforementioned is a key principle. (p. 472)

Throughout this thesis the MVC will be considered a design pattern, rather than an available framework.

2.7 Temporal data

The word ‘temporal’ stems from ‘temporary’, and its meaning is also the same; not eternal. (Princeton:WordNet)

In the context of temporal databases, Farlex (2005) defines temporal data as data which depends on time in some way. Similarly, Gregory (2002) terms temporal data as data that explicitly refer to time.

In the setting of media presentation and compression, Fernanda and Ebrahimi (2002) describe temporal information, or data, as the point in a timeline when an mpeg-4 object starts its playback procedure.

In the field of artificial intelligence, temporal logics enable expressions to be quantifiable with regard to time, enabling such strategies as Reinforcement Learning where agents learn ‘better’ by multiple attempts over time. (Luger 2002)

In this thesis video sequences are regarded as temporal data, because I want to take the passing of time into consideration in digital, interactive art.

2.8 State machines

State machines are objects that contain a set of input and output events, a set of states, a function that map states and input to output, a function that maps states and input to states, and a description of the initial state of the machine. (Selic, Gullekson et al. 1994)

There are many forms of state machines and considerable literature available on the subject. The subject is only related to the implementation of the case installation however. State machines are not of enough relevance to the research questions or research findings to warrant a more thorough review.

Technologies of digital, interactive art

Existing technologies used in digital, interactive art can be divided into two main categories; hardware and software. This chapter covers the most relevant aspects of hardware, and the software options that exist in the realm of Java for handling audiovisual material such as movies.

3.1 Hardware

There are three major technological paradigms of hardware within interactive art that traditionally defined the genre throughout its growth and development. They all date back some fifty years, to the participational art forms of the late sixties. (Dinkla 1994) According to Dinkla these are responsive environments, real-time manipulated video and head-mounted display. This section covers these three, the usage of the head-mounted display in virtual and augmented realities, and finally offers a brief description of the most common sensory devices used to collect audience input.

Responsive Environments

Arguably, the first contribution to interactive art controlled by a computer, was Myron Kreuger's installation; Glowflow. (Hieronymi 2004) This art piece was displayed in a museum, physically an entire room lit by glowing tubes of phosphorous liquid. The room had six loudspeakers, emitting sound generated by the computer that also controlled the lights. The computer altered the audiovisual appearance of the room based on sensors in the floor, detecting at all times where the audience was located. Dinkla defines this sort of installation, where the physical surroundings react to the audience as a responsive environment.

Real-time manipulation of video

After Glowspace, Kruger adopted a different approach to the technical aspects of his art form. In Video art the use of cameras reflecting to the viewer an image of themselves, had began to grow. Kruger combined this idea with a computer, effectively enabling a near infinite number of possible transformations.

In 1974 he finished the first version of his work 'Videoplace'. In this installation, which he continued to develop new instances of, viewers are confronted with a video-image of themselves, projected onto a canvas by a computer. (Hieronymi 2004) The computer manipulates the image in several different ways, adding transformations or new data. The perhaps most famous of these instantiations is the 'Critter', a green figure that is added to the image, crawling up the outline of the viewer. Once it reaches the top of the outline, it performs a celebration dance. This causes an interesting effect for the viewer, because they can change their own position, effectively altering what is the highest point of their

body. A feeling of control over the Critter and thus the system is given to the viewers further pushing them away from the passive observer. (Dinkla 1994)

Several other works of interactive art using similar technology were soon following, not only from Kruger. One strategy involves the camera capturing the result of its own, live video stream of the viewer. The delay between the input and output signals combined with the seemingly infinite number of recaptures of every previous image, created a dazzling effect.

Head-mounted display (Virtual reality)

More or less in parallel with the use of real-time video, another branch of technology was developed and put in use by digital artists. Popularly called *virtual reality goggles*, the head-mounted display is basically a computer screen positioned on the inside of a helmet, a pair of glasses or a similar device.

An image of a digital world, produced by a computer is then displayed on the device, giving an illusion of being in a totally different reality. The device is equipped with sensors capable of telling its location and heading in its environment, enabling the computer to tell where the person is, its heading and movement. Virtual reality goggles were considered the next big thing some within computer games some years back, as illustrated in figure 5.

Figure 5 Head-mounted display

For a while these gained a lot of popularity not only within the digital art community. Computer games, remote medical operations and remotely controlled vehicles are among some of the projected use this technology was thought to cover. (Dinkla 1994) The basic idea was that the computer could create any audiovisual environment, replicate it through the device head-mounted display and respond to data from input-devices such as gloves or boots. Later on full suits were made, adding pressure points to simulate touch.

Augmented reality

Augmented reality differs from Virtual reality in that the aim is not to create an entirely new reality, but to reconcile ours with the computer world by adding computer generated images, animations, sounds to physical objects. (Wellner, Mackay et al. 1993)

Augmented reality is a growing area, already being used in fields such as academics, medicine, entertainment, military training, engineering design, robotics and telerobotics, manufacturing and consumer design. Figure 6 shows augmented reality used for nautical navigation.

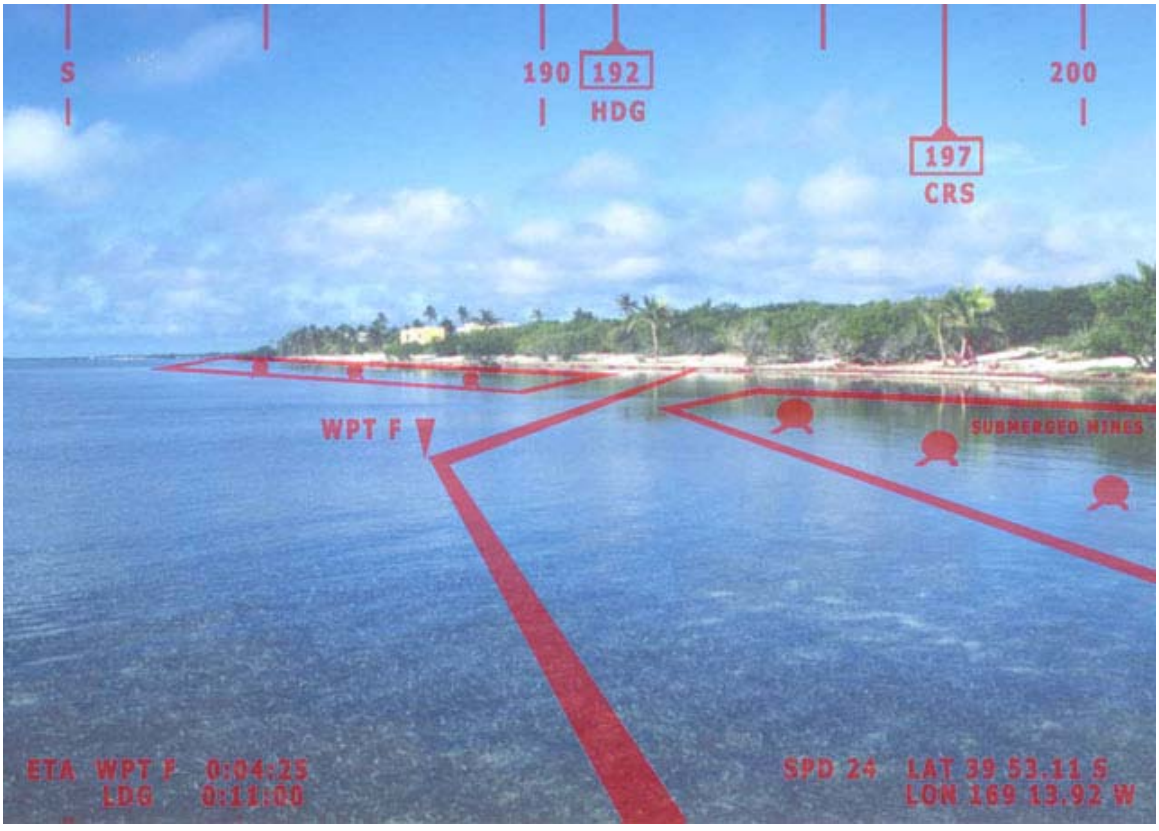

Figure 6 Augmented reality

Sensory devices

Vital in any form of interactive art whether the audience is using head-mounted displays or watching manipulated video, is the hardware responsible for registering input from the installation to the audience. Typical devices such as mice, keyboards or joysticks are obvious examples of sensory equipment with which we are familiar. Other equipment used by installations at Ars Electronica included cameras, pressure-pads, microphones, GPS-devices and cellular phones.

3.2 Java software solutions for handling media

Java is not the only software technology capable of performing playback of media; in fact it might be among the less likely candidates a computer scientist might think of when asked to manipulate audiovisual data.

Applications made in Java are considered slow, thus other programming languages such as C are often preferred. Additionally, applications made for the specific purpose of real-time manipulation; playback and capture of audiovisual data such as MSP/Jitter are commonly in use. Java has many other qualities however, and a critical examination of its potential in interactive digital art is one of the two main subjects of this thesis.

Interactive art may, as other art, be exhibited at different museums, public locations and more. Different from other forms of museum-exhibited art however, is the fact that installations of modern interactive art can be displayed several places simultaneously, due to its digital content. The Mona Lisa can only be at one place at any given time; copies will never be able to attract the same audience as the original painting. For digital art, this is different.

When entering a new museum, public space or other exhibition areas, several elements of the installation may change. This includes sensory hardware, computer hardware, operating system, software version revisions etc. To facilitate an easy transition between these different specifications, a robust and platform-independent system is required.

The Java programming language claims to deliver both of these qualities, as well as several other benefits. It is object oriented and supports all of the commonly associated ideas; heritage, variable scopes etc. Java is fairly easy to learn, and is now the first programming language offered by most educational institutions. Building on the Java platform gives the developer access to the myriad of extensions and plug-ins to the language which are available through the net. Finally, Java is free of cost, and in its most basic form requires nothing more than what is offered as a simple download on Sun's website www.java.com.

This makes Java seem like a good implementation-platform, especially when it comes to the business logic and control-application of an installation. Interactive art, however, requires more than just the logic behind. Sound, images, compressed and uncompressed video, as well as three-dimensional graphics and other complex data structures must all be computed, transformed and displayed – often in real-time. This adds a new level of requirement, namely speed.

When compiling a file with Java source code, the resulting file is not an executable program but a 'classfile'. This file contains a set of 'bytecodes', which is a more machine-friendly representation of the original source-code. This file can then be interpreted real-time on the target machine and/or platform by a Virtual Machine. (Deitel and Deitel 1999)

Every platform that has a Virtual Machine will be able to decode the classfile, effectively making Java platform-independent. Some limits exist on certain platforms, for instance J2ME (Java 2 Platform, Micro Edition) lacks support for floating point numbers, trigonometric functions and more due to the limited hardware of mobile devices such as cell phones.

The catch to the Virtual Machine approach is obvious; adding the job of real-time compiling to the aforementioned processing of audio and visual data causes serious performance degradations. (Deitel and Deitel 1999)

This raises the question of whether Java is capable of handling several forms of media simultaneously, delivering adequate and stable performance. Additionally, Java is traditionally a web and database-application language, making the support for presentation and manipulation of heavy media such as video sparse.

I will in the following subchapters describe the different Java-related technologies available in this category, provide a break-down of what they offer, and finally show how Java in fact is able to provide the necessary speed and media-support required for installations of interactive art.

The technologies covered are Java Media Framework, IBM Toolkit for MPEG-4 and Quicktime for Java. Of the three, the Java Media Framework will be explored in the most detail as it is used in the Tapet installation.

Java Media Framework (JMF)

This section describing the Java Media Framework offers a brief walkthrough of its history and architecture, a description of the use and relevance of its major classes, and finally a description of the possibilities of extending the framework.

History

The original Java specifications from version 1.0 in 1995 up to the most recent 1.5 include little or no support for handling advanced media content such as video or sound. In later revisions Java2D, Java3D, Java Sound, Java Speech, Java Telephony and Java Advanced Imaging have been added to deal with their respective areas, as well as the Java Media Framework.[6]

Mid-1998 the first version of the JMF API, 1.0, was released. It had been designed and specified by the trio Sun, Silicon Graphics (SGI) and Intel. Shortly Both SGI and Intel withdrew from the project after this release, leaving it in a state of stalemate. Shortly after, at the end of 1998, IBM showed an interest and joined the JMF project.

With the help of IBM, JMF 1.1 was released in December the same year. This version was an all-Java implementation, meaning it could run on every Java compliant platform. In addition, native implementations for the Win32 and Solaris platforms were released. These supported more media types, and due to their native libraries offered much better performance. The all-Java version however, gives full platform-independence (Gordon and Talley 1999) which made it ideal for handheld devices and applets run in web-browsers.
IBM has since abandoned the project, again leaving Sun as the sole benefactor of JMF. The most recent update is per today 2.1.1e, and is available through Sun's website at the following URL; http://java.sun.com/products/java-media/jmf/.

Architecture

A good comparison of the Java Media Framework architecture is that of a stereo. (Kurniawan 2001)

Roughly said, both can be divided into three main parts. For the stereo these are;

- The input stage; music is read from the radio network, a CD, tape or similar.
- The processing stage; the music is processed by the amplifier and prepared for deliverance to the final stage.

---

[6] Gordon and Talley (1999)Gordon, R. and S. Talley (1999). Essential JMF. Upper Saddle River, New Jersey, Prentice Hall.
(p. 6)

- The output stage; the finished music is delivered to the audience through a set of speakers, an earphone or a similar output device.

JMF can be divided into the same three stages as shown in figure 7. (www.grack.com 1998)

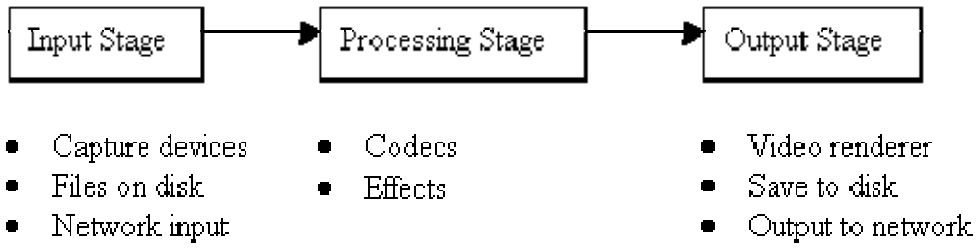

Figure 7 Java Media Framework overview

Main components/classes

JMF is composed of a myriad of classes and interfaces, some obscure and rarely used, some necessary for every media-enabled JMF application or applet. The lifecycle of the most basic JMF application or applet starts with a DataSource, a media-container that can read from any form of media input. Once the DataSource has been established, a Manager can create a Player or Processor from the DataSource, depending on the needs of the program. The input media will be of a given Format, which needs to be decoded by the proper Codec before being fed to either a DataSink or Renderer as final output.

DataSource

Just like a stereo can have different input-devices delivering music, be it a CD, radio or a tape, so can JMF. These come in the form of network streams, capture devices, remote or local files containing media. JMF calls these DataSources. The DataSource contains both the location, the protocol used to transfer and the necessary software to deliver the media.

There are two main versions of the DataSource, they are classified by how data-transfer is initiated between the actual source, and its destination.

The PullDataSource is the most common, where a client requests data, and receives it promptly. Stored or remote files read by a media-player application are common examples of this.

PushDataSource works in the opposite way; the server initiates, and determines content and dataflow. Broadcast media such as television fits this description.

Manager

The Manager is a class used primarily for interfacing other existing classes, such as creating a Player from a DataSource, or adding a new Codec. There are four managers, the Manager, CaptureDeviceManager, PackageManager and PlugInManager.

The base Manager class is used to create and interface DataSources, Players, Processors and DataSinks. Given the URI of a file it can, for instance, create a DataSource instance for that specific media file.

The CaptureDeviceManager holds the list of registered capture devices such as cameras, microphones and webcams. Access to these is granted through the CaptureDeviceManager, as well as the possibility to add and remove items from the list. PackageManager is the equivalent for packages containing custom Players, Processors or other classes to be used by the JMF. Finally, the PlugInManager maintains a register of plug-in classes that do intermediary processes to the media data.

Player

The Player class can be created from pretty much any input source, and is used to play through one or more tracks of DataSources. These are output to either a graphical display or a sound adapter.

The Player extends the class Controller, which in turn extends Clock. This makes Player an event-driven class controlled by time. Events are triggered at several points in time of a Player object’s lifetime.

The first events that occur are during the realization and prefetching of the Player. This is when parts of the media to be played is pre-cached to minimize the chance of chopped-up sound, missing frames or other artifacts that occur when the available processing power of the hardware proves insufficient.

The six states of the Player that trigger events are, chronologically, as follows:

- Unrealized: In this state, the Player object has been instantiated. A newly instantiated Player does not know anything about its media.
- Realizing: A Player moves from the unrealized state to the realizing state when you call the Player's realize() method. In the realizing state, the Player is in the process of determining its resource requirements. A realizing Player often downloads assets over the network.
- Realized: Transitioning from the realizing state, the Player comes into the realized state. In this state the Player knows what resources it needs and has information about the type of media it is to present. It can also provide visual components and controls, and its connections to other objects in the system are in place.
- Prefetching: When the prefetch() method is called, a Player moves from the realized state into the prefetching state. A prefetching Player is preparing to present its media. During this phase, the Player preloads data, obtains exclusive-use resources, and does whatever else is needed to play the media data.

- Prefetched: The state where the Player has finished prefetching media data - it's ready to start.
- Started: This state is entered when you call the start() method. The Player has started presening the media data.

(Kurniawan 2001)

In reality, the states “realizing” and “prefetching” are triggered several times, while the Player is preparing to start playback of its media. The Player will trigger other events as well, on other grounds than state-changes. The method “setMediaTime()” gives the Player a new position in the DataSource to commence playback from. When called, this and similar methods will trigger events that can be handled by the parent application. There is also a special event that occurs when the playback is finished, called “EndOfMediaEvent”. A common use of this event is to set the media time to zero, and restart the Player. In effect, this will put the Player on auto-loop, never ending until requested by the user/client.

Processor

The class Processor extends Player, meaning it inherits all of the Player’s methods, playback capabilities, states and events. Additionally, Processor offers increased opportunities for controlling what operations and processing is performed on the media. This is achieved by providing two additional states:

- Configuring: A Processor enters the configuring state from the unrealized state when the configure() method is called. A Processor exists in the configuring state when it connects to the DataSource, demultiplexes the input stream, and accesses information about the format of the input data.
- Configured: From the configuring state, a Processor moves into the configured state when it is connected to the DataSource and the data format has been determined.

Figure 8 gives a good idea of how the Processor can be used to manipulate media real-time. A DataSource with two input-tracks, audio and video, goes through the demultiplexer and is split. Plug-ins can at this access the encoded material. A codec plug-in will then decompress the data, and plug-ins that access raw data can be applied. Finally the finished media data is output either through a renderer for each track, or through the multiplexer which will combine the two streams into one, final output DataSource.

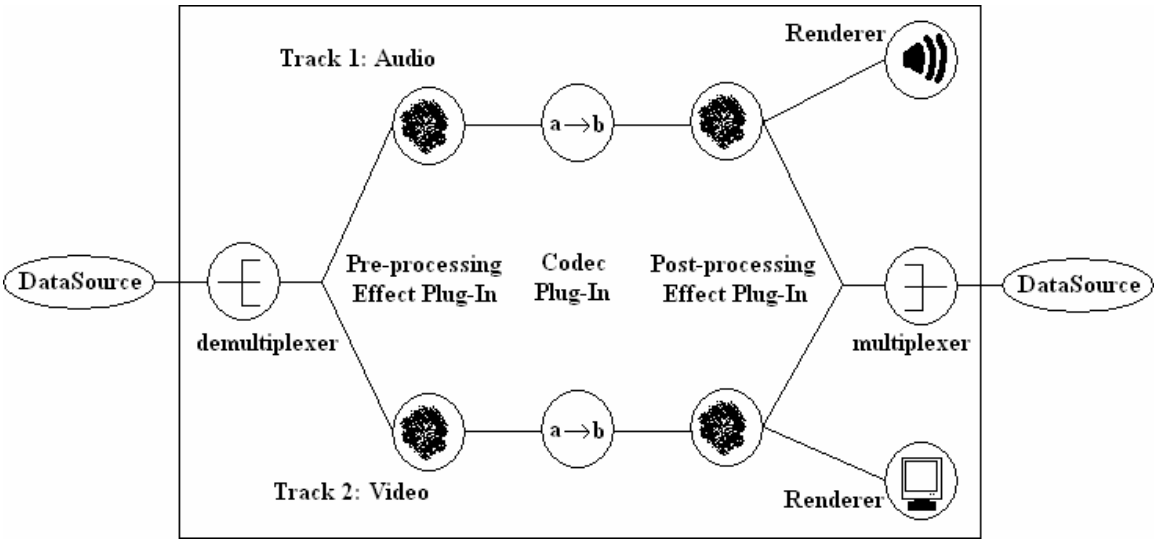

Figure 8 JMF Processor architecture

Format

Every track of audio or visual data contained by a DataSource, is of a given Format. The Format class does not contain the necessary software to decode the data, but is a descriptor-class, providing detailed information about the encoding used, bitrate of audio etc. Format has two direct subclasses, AudioFormat and VideoFormat. VideoFormat again has six pre-defined subclasses:

H261Format, H263Format, IndexedColorFormat, JPEGFormat, RGBFormat and YUVFormat.

A format must be defined, both for the incoming and outgoing data of a track or plug-in.

Codec

A codec takes an input stream of one of its specified, supported input formats, and transforms it into a different format before returning it. While the codecs supplied with JMF cover the most basic formats of both video and audio data, these are not sufficient with the growth of newer, better formats such as DIVX or MPEG-4.

Effect

An effect is much like a codec, in that it takes an input stream, transforms and returns it. The difference lies in the transformation performed; while a codec changes the format of the stream, leaving the actual data to be presented untouched, an effect does the opposite. The stream returned is of the same format, but its content has been processed by the effect. This could be as simple as registering the average color of the different images in a video-track, or it could apply a noise-reduction algorithm to classical music. JMF does not come with pre-defined effects; there are a few examples on Sun's website that can be

used. However, the most interesting and useful ones must be coded and added manually by the programmer.

DataSink

A DataSink object is used to output the resulting media after applying codec and effect transformations to non-standard destinations, such as a file in the form of a file-writer, or forwarding the result from one Processor to another.

Renderer

The default destination for both Players and Processors is a renderer. A renderer draws visual data and performs playback of audio. There are several different renderers available, as well as the possibility to add new, custom-made renderers. Some of the projects providing codec-plug-ins earlier provide custom renderers, which in some cases perform better than the ones supplied with JMF.

Extending JMF

As mentioned before, the capabilities of JMF when it comes to supported media types, performance of playback and allowed transport layers is decent, but not good enough to properly cover the needs of modern applications. Fortunately, Sun foresaw this issue when they designed the JMF 2.0 specifications, and thus the architecture of JMF allows for extensions to be added to all of these fields.

The most notable of these extension-capabilities, named plug-ins, includes MediaHandlers, Codecs, Effects and DataSources. Writing these extensions can be time-consuming and tedious. Fortunately several projects exist, where extensions to JMF are being developed for use by the JMF community.

This section will provide an overview of the most useful projects related to increasing the range of media support in JMF; FOBS, JFFMPEG and IBM's MPEG-4 for JMF.

FOBS

"FOBS" is an open source project hosted by 'Sourceforge', that offers a lot of functionality both to the C++ environment, and JMF. Unlike other open source projects, on most platforms it comes with a binary installer, making it far easier to put in use.

FOBS is based on the open source project "ffmpeg", and offers some of its functionality to both target platforms (C++ and JMF).

> *" **FFmpeg** is a complete solution to record, convert and stream audio and video. It includes **libavcodec**, the leading audio/video codec library. FFmpeg is developed under Linux, but it can be compiled on most operating systems, including Windows."* (Sourceforge 2004)[7]

Unfortunately, Ffmpeg is currently threatened by the new legislation regarding copyright that is under consideration in the European Parliament. Many of the supported codecs are protected by copyright laws and patents in America, should the suggested laws be constituted the same will apply in the European Union.

JFFMPEG

Much like FOBS, "Jffmpeg" is an open source project, built on the Ffmpeg project and hosted by sourceforge. Slightly smaller in scope, Jffmpeg is a pure codec plug-in while FOBS cover other areas such as renderers. Jffmpeg adds several additional formats

[7] Description of Ffmpeg on their webpage at sourceforge. Essentially Ffmpeg is an open source solution to all issues related to audiovisual playback, streaming and recording of multiple formats.

however, and is therefore a good addition to the Java Media Framework, with or without FOBS.

Most notably the file formats “VOB” and “OGG” are supported in pure Java versions. Additionally, several audio and video formats are supported in pure Java, while many more can be accessed directly from Ffmpeg through the Java Native Interface.

IBM MPEG4 for JMF

When IBM took the place of Intel and Silicon Graphics in developing JMF 2.0, they gave this job to the Haifa Research Labs in Israel, a part of IBM’s Alphaworks division. Alphaworks soon discovered that a major flaw with the new JMF, was its lack of support for the up-and-coming new media-compression format; mpeg-4.

In 2002 a pure Java version of a pure mpeg-4 codec plug-in was finished. The plug-in appears quite efficient, and their website claims an even more efficient native version that is available upon request.

The codec adds an IBM logo to the top-left part of the screen when decoding video. For obvious reasons this is not ideal in pretty much any setting, and is a result of the downloadable version not being licensed. In the FAQ on Alphaworks website it is stated that the licensed version is not yet available, but hopefully soon will be.

The decoder is fully compliant with the MPEG-4 Visual (ISO/IEC 14496-2), Version 2, Simple Profile. RTP modes D and F are supported as described in RFC3016.
There is also an encoder, but due to legal issues with the mpeg-4 encoding algorithms it cannot be released. (IBM 2002)

IBM Toolkit for MPEG-4

In parallel to the mpeg-4 for JMF plug-in, IBM launched a project called the “MPEG-4 Toolkit”. It is essentially a set of Java classes, designed specifically for the purpose of creating and displaying mpeg-4 content.

Accompanying the actual Java classes, are five sample applications that are built on the toolkit. These include tools for creating mpeg-4 audio and video content, a tool for creating mpeg-4 content beyond simple audio and video files and three different mpeg-4 media players. These come in the form of one application, and two applets. One of which reads local media files, and one that reads over the HTTP protocol.

The Toolkit is the only Java alternative I have found, that can produce or perform playback of complex mpeg-4 media. That is, content that goes beyond a file containing a single track of video and/or a single track of audio.

The packaged applications are able to combine already existing mpeg-4 media into more complex forms, but much like with IBM's JMF plug-in, the actual encoding of mpeg-4 data is under copyright, and not available for publication.

The decoding of mpeg-4 Simple Profile video is done with a modified version of the "MPEG-4 for JMF" plug-in to JMF. The list of supported formats is as follows;

- MP4: ISMA-compliant
- MP4: including MPEG-4 Systems
- AVI: MPEG-4 Simple Profile video (such as Divx, Xvid) with MP3 audio.
- MPEG-4 Simple Profile video (.cmp, .m4v, .263)
- AAC Low-Complexity Profile audio (.aac, .adif, .adts)
- MP3 MPEG-1 and MPEG-2 Layer 3 audio (.mp3)

(IBM 2003)

Much like with JMF, the playback is controlled by creating a Player object, and giving it an input media file or network stream. Unfortunately, there is no equivalent of the Processor class provided by JMF. This considerably limits the possibility of adding custom effects and transformations to the media, even though some basic functions such as transparency are a part of the mpeg-4 specification.

Quicktime for Java (QFJ)

The only real alternative to JMF is Apple's Quicktime for Java, termed "the rival API" of JMF. (Adamson 2002)

This section will cover the basic architecture employed by QFJ, an examination of its abilities to handle image manipulation beyond the provided methods and finally a comparison to the JMF.

Architecture

The base structure and architecture of Quicktime for Java is considerably different from that of other Java APIs. There are a vast number of classes and packages, of which a large number must be imported for any application utilizing QFJ. (Adamson 2003) According to Adamson, this is in large due to the difference in implementation order; a regular Java API is developed, followed by a reference implementation. JFQ already had the reference implementation written in C, and the Java API was written to fit the existing implementation. With the C implementation being procedure-based, the object to object mapping in the API was largely done by collecting the C-procedures that took arguments of the same type, adding them as methods to an object of that type in Java.

QFJ is, as most Java 2 and later API's based on the MVC pattern to separate its model from presentation and controllers. (Stewart 1999)

As JMF, QFJ is thread-based. At creation, a movie thread is given extremely low priority meaning its amount of processor time is highly limited. In the original C library a programmer must repeatedly call the moviesTask function with a movie thread as argument to give it processor time. In the Java API this is largely taken care of by a separate thread calling the task() method regularly on all open movie objects. The only exceptions are objects that have not yet been added to any form of GUI. Getting details such as format or length of these can only be achieved by calling task() repeatedly till the needed details have been loaded.

Another interesting feature of QFJ is the way it handles important media events such as end and changes in framerate. In the JMF a listener is set up which will trigger an exception whenever an event occurs. QFJ offers the ability to prepare a *callback* object, which given a movie reference and an event will perform one single run of the execute method. While similar to the approach with a listener, it simply inserts a method-call at the appointed time rather than constantly listening, thus freeing some processor time. (Adamson 2003)

Extracting raw image data

Through QFJ's impressive number of packages and classes, a lot of useful tools are provided such as extracting parts of a movie, changing audio tracks and much more. However, when manipulation of either video or audio tracks beyond that of the supplied methods is needed, the raw data must be accessible in an efficient manner.

Davison (2005) adds a movie clip to the 3d chess world taken from his book "Killer Game Programming in Java". (Davison 2005) Using the MVC as a pattern to distinguish the view from the controller and the model, he was able to easily make two different versions; one using the JMF for the model component, and one using QFJ.

Using the video as texture for surfaces drawn using the new Java3D API, the output data of the model must be of

> "*the TYPE_3BYTE_BGR BufferedImage format, which is necessary for the Java 3D parts of the program to employ texturing by reference.*" (Davison 2005) (p. 2)

In the JMF version he could set the output format directly to what he needed. QFJ however, outputs Apple's PICT format from 1984 when requested for single images. (Leurs 2000)

Searching for efficient ways to convert PICT to TYPE_3BYTE_BGR BufferedImage, Davison found two; the first on a thread about image extraction from QuickTime movies on the QuickTime for Java mailing list (QTF-mailing-list 2002) which involves creating a QuickTime Graphic object, converting this to a raw image and finally iterating through the pixels of this raw image, converting the byte-order to match the needed format in the BufferedImage.

The second comes from the standard book on QFJ (Davison article 2005) “QuickTime for Java: A Developers Notebook” (Adamson 2005) and involves adding a dummy 512 byte header to the PICT image, letting Quicktime’s ImageProducer think it is an actual PICT file. The ImageProducer will output a standard Java Image object, from which pixels can be extracted.

QFJ vs JMF

One of the features of Java compared to other programming languages, is the platform-independence; being able to run your programs on all platforms without recompiling or changing any code. It is a novel idea, and while it to some extent works better in theory than practice, it is possible also with digital media.

Adamson (2002) shows how to build a Jar archive that contains, decodes and plays the media on its own, regardless of platform. For this example Adamson uses JMF, because there is no pure Java version of QFJ. Additionally, native versions only exist for Mac and Windows systems, thus no support for Linux or other operating systems, while JMF has an all-Java version, as well as native versions for enhanced performance. (Adamson 2003)

The range of supported media types is an important aspect of both JMF and QFJ. The media supported by JMF is highly limited, especially newer codecs such as MPEG-4 and H264 are sorely missing.

QFJ supports all of these and more, because it builds on the newest Quicktime libraries which are continuously updated. Increasing the number of supported media types for JMF is possible. Due to the open architecture new codecs, rendering components and more can be added. There are several open source projects that take advantage of this as mentioned earlier in this chapter. It is even possible to make a bridge between JMF and QFJ, effectively opening the huge media support of Quicktime to JMF as well. (Adamson 2002)

During tests of the aforementioned 3D chess project, both QFJ solutions proved considerably slower than the JMF alternative. The major reason for the difference in performance lies in the conversion procedure. While JMF allows adding the specification of output format to the codec chain, QFJ forces the programmer to convert the output manually, adding considerable processor overhead.

In conclusion, the wider range of supported codecs and ease of use offered in most common operations such as playback, simple movie editing and mixing of tracks makes QFJ preferable to JMF in ‘normal’ applications. This includes media players, video editing programs and similar.

Faced however, with more complex operations that require custom manipulation of raw image or audio data the overhead in converting from proprietary Apple formats causes

considerable computational overhead, making JMF the preferable choice. Additionally QFJ is restricted to the Mac and Windows platforms forcing applications that require platform-independence to choose JMF.

4 Research questions

The study and creation of digital interactive art requires a mix of several professions. Besides the obviously needed artistic element, the actual technical implementation of such installations may require several forms of specialist knowledge or skill. Reflecting this, the research questions of this thesis originate from two different problems we encountered during the creation of Tapet;

- The technical design of the Tapet application would require several pieces of software, but we were uncertain as to what platforms to base these on.
- Our project staff consisted of individuals with different professional backgrounds. We needed a conceptual framework we could use when designing, evaluating, reflecting on and discussing the interactivity of the installation

This chapter will describe the two research questions built on the aforementioned problems and the methods utilized in solving the problems and developing the case project: Tapet.

4.1 Java as implementation platform in interactive art

During the very initial phase of development and planning, finding available technologies and deciding which were the most suited to our project and limitations was a vital issue. Due to Java being the programming language and platform I was the most familiar with from earlier studies and own practice, exploring whether it was useful in this setting was an early decision.

What are the implications of using Java technology as development platform for an installation of interactive art?
More specifically, the advantages or disadvantages connected to using Java for

a) the underlying business logic layer
b) audiovisual presentation layer in the form of video with sound
c) a communication layer, synchronizing the two other layers

During my visit to Ars Electronica in Austria, it became apparent that modern installations of interactive art almost exclusively have some sort of computer program performing one or more of the main tasks involved. The software platforms these computers ran were usually programs specifically designed for this purpose, such as MAX/MSP with Jitter. Some installations had software pieces programmed in C or C++, reviewing the older ones in the “library” revealed that some even contained methods coded in pure assembly language, but I found no use of Java in this context.

As installations of interactive art have grown in both size and complexity, the need for technical expertise has grown. Ideally the artist would need only to consider the artistic aspect of an installation, while the technical details are worked out by specialists.

Programs specifically designed for interactive art do a good job at most functions, but are in nature limited to whatever functionality and metaphors they contain. While it is possible to extend these programs, as well as find solutions to problems that could arise outside of the software domain, I argue that in order to gain more freedom of choice over content and presentation, applications will often need to be specifically designed for the purpose of each installation.

It is in this setting the use of Java as implementation platform would be of interest. Any who have studied computer science (recently) have inevitably used Java to some extent; Java is among the most common programming languages taught by nearly every educational institution offering a Computer Science degree. (Tyma 1998)

Java is also available for most platforms including small devices, allowing more freedom of choice in deployment type and platform. (Tolba, Briceño et al. 1998)

4.2 Conceptual framework

Working with specialists from several professions, mostly in tandem with a media artist, required some sort of common language that could describe how the system responds to the actions of the audience.

What concepts or models supports the process of describing the aspects of the interacting that is carried out by the system of an interactive art installation?
In particular I will address the issues of the narrative structure with respect to time and location.

As shown in section two, much work has been spent defining and describing interactivity and its effects. Lee (2000) developed a model for observing interactivity, based on user perception and interaction setting. While Lee's and similar models have been applied to settings such as graphical user interface components, I wish to apply Lee's model to the setting of digital, interactive art.

As the word implies, interaction requires some form of information exchange. In the setting of an interactive installation, the part provided by the audience can only be guided to a certain extent by signs, markings or similar. It is the other half, the reactions of the system to whatever input the audience provides, which is entirely up to the artist.

Given the importance of the responsiveness of an installation, having some sort of system in which the different kinds of reaction is organized, as well as the major effects they have on user experience, should help the design process.

By applying Lee's model it is my ambition to structure the feedback to user input in the interactive art setting in a way that can be reproduced, and used in the design process of installations similar to that of the case study Tapet.

4.3 Method of research

The overall method of research employed to answer the research questions involved the implementation of a case installation; Tapet, literature review, field study and the use of software engineering principles. The development process of the case installation was highly dependant on the other aspects of research.

This section contains a brief discussion of qualitative versus quantitative research, a brief explanation of the case installation and a description of the overall method adopted in designing and developing the Tapet system. The latter part of the section also contains more detailed information on the literature review, field study and other aspects of method employed also in the answering of the research questions.

Case implementation

A major part in the research leading up to this thesis was the fulfillment of the development cycle of an installation of digital, interactive art called “Tapet.”

Tapet went from being a highly complex installation of interactive art with a core consisting of software that I was to design, to being a considerable software engineering project with several application-class modules running the interaction, sound, video and control components of the installation. The project was developed in a research-setting at the research centre Intermedia at the University of Oslo, thus constituting a high degree of experimental development.

The development of the installation required considerable algorithmic insights in subjects such as chroma keying, anti-aliasing and other real-time audiovisual manipulation methods.

Qualitative versus quantitative research

The differences between qualitative and quantitative research methods are well known; where quantitative studies gather a considerable number of items that are of easily quantifiable aspects, qualitative studies tend to go into more detail with each item due to aspects that are less easy to quantify. The advantages of each type of research are a perennial, hot debate. (Neill 2004)

Still, certain results of choosing either method are generally accepted; data from quantitative research are well suited for statistical results and tend to be easier to generalize. Data from qualitative research can go deeper into their subject, and are better suited for studies where the researcher does not know fully what he or she is looking for.

For the purposes of this thesis, qualitative research was preferable for a number of reasons. The need for a detailed description of each test-participant’s perception of interactivity in the different settings called for a qualitative approach. The nature of the case installation used for the interviews implies that each person has a very different

experience. This means that creating a set of questions that can be used quantitatively without loosing much of the disparity as experienced by the audience is difficult. Finally, pure practical reasons such as the availability of the studio where the installation was set up limited the amount of test participants that could be interviewed. With a very limited set of audience, presenting the data as a statistical outcome could not be justified.

Method of development

The overall method of research as well as implementation and installation of Tapet was a process divided into three main components each following a cycle of learning. (Smørdal and Kaasbøll 1996)

In terms of software engineering the method of development has the most in common with Agile Software Development, a

> "*Conceptual framework for undertaking software engineering projects.*" (Wikipedia 2006)

Agile software engineering is a design architecture similar to that of spiraling, but with shorter iterations. Face to face communication is favored rather than paperwork when reviewing the different phases of each iteration.

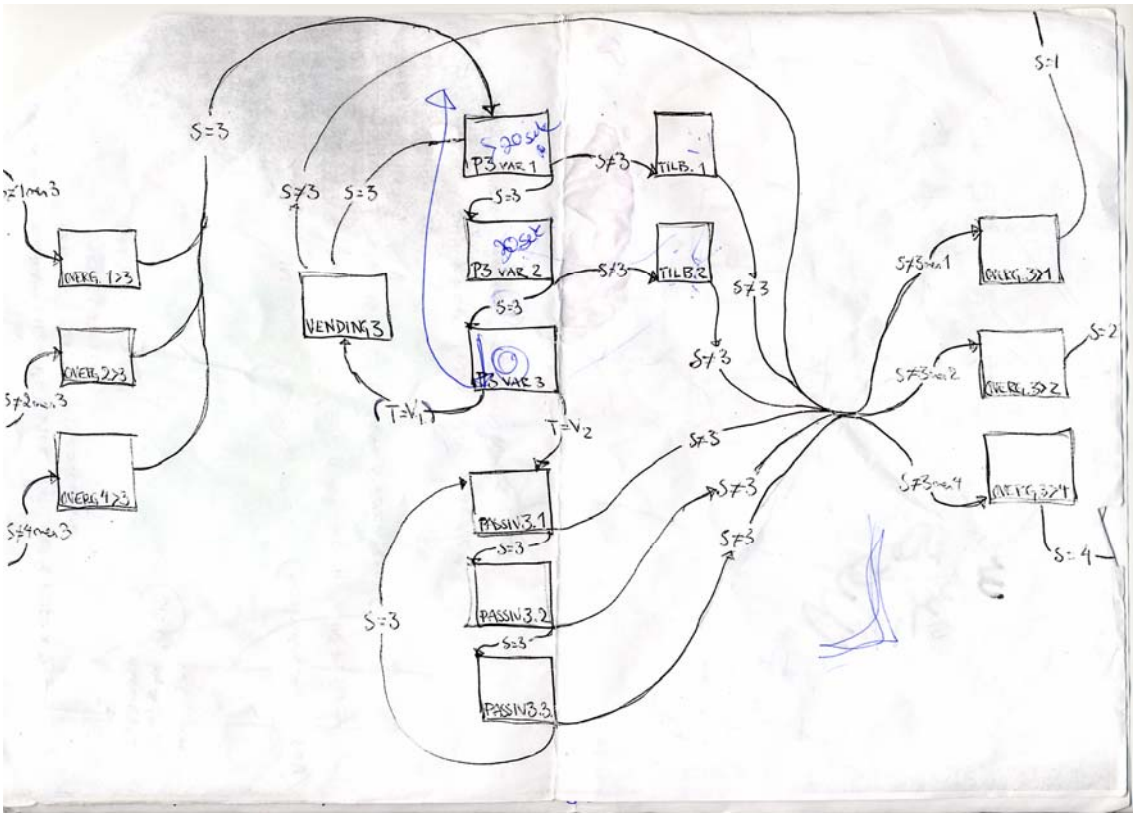

Figure 9 Early state chart diagram

Other software engineering practices were used as well. Figure 9 shows the original state chart diagram, listing the different states with their possible transitions forming the specification of a state machine. Position P in the diagram is specified by the previous position, the time T and sensory input S.

Data and knowledge gathered through the literature review and field study were used as input for a design component. Software design patterns were used both for the design and the implementation process. The results of the design were then used in implementing a working prototype, which was tested through observed use. The results of the test process were then examined and analyzed, before another iteration of design was initiated with the knowledge gained from the previous round.

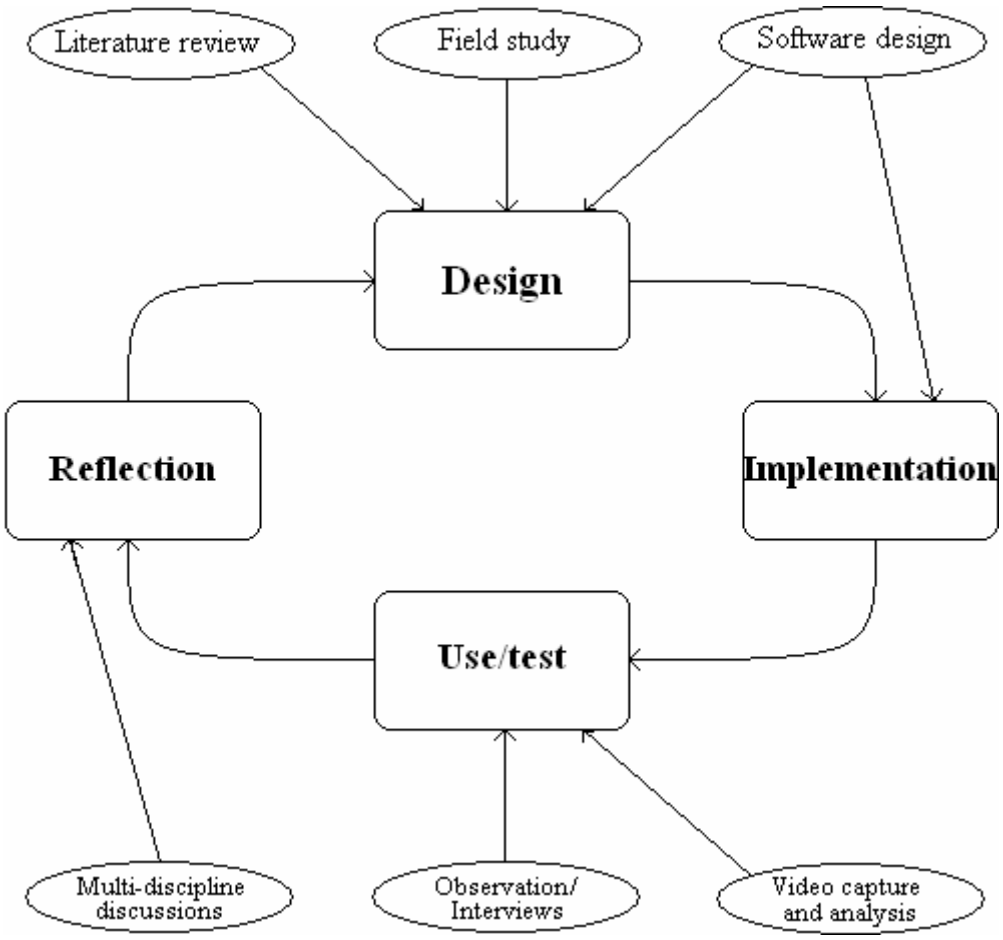

Figure 10 Learning cycle

This circular development process was used throughout the development process of Tapet, with main focus on different aspects of the circle in each iteration.

In the early stages of the process, technical design, the implementation was only used to see what was actually *possible*. With limited funding and time, starting development with a technology that later proved incapable would be fatal.

During implementation of the software, tests were done with other members of the team, comparing possible input and output video formats, testing what framerate would have to be achieved and more. Finally, adjusting the narrative structure focused mostly on the test and reflection side of the circle.

The iterations ranged from one week to a full month, usually ending in a meeting where multi-discipline discussions would decide the scope of the next iteration.

In the next few sections, the relevance of each component in the learning cycle will be explained.

Design

The technical design of the installation was based on three main sources of input besides experimentation; software engineering (design) principles, a field study and literature review; each covered in the following sections.

Software design/engineering

Software engineering principles were used in both the design and implementation components of the process, utilizing such tools as state-chart diagrams and UML class diagrams.

With a fair amount of uncertainty related to the components of the installation not yet implemented, an open architecture and ease of modification was maintained through choice of implementation platform, communication protocols and a high level of abstraction.

Literature review

An extensive review of existing literature on the topics relevant to the thesis was conducted. The subjects covered included interactivity theory, architectural and design patterns, embodied interaction, narrative structures, video encoding/decoding theory and java related media technologies.

Field study

After joining the Tapet team, I was fortunate enough to be among a group of four people that traveled on a field trip to Linz in Austria, to the 25th Ars Electronica festival. I was included on this trip to learn more about what digital interactive art is all about, easing the understanding of the system that I was about to build.

During the visit we saw what I believe must be some of the best examples of what this type of art has to offer, as well as a review of the contributions from earlier years in the festival library.

Talking to the artists and designers that presented their installations gave a good idea of how much work is actually involved when producing this kind of art, as well as a notion of how.

Talks were given by among others Bruce Sterling and Krzysztof Wodiczko on topics related to the theme of the festival – Timeshift; with the festival in its 25th year, what would the following 25 years bring to digital art and the digital world?

Implementation

The implementation of the adopted software architecture was done in Java using the Java Media Framework API available from Sun. Software engineering principles were, of course, important also in this phase of the learning cycle.

Use/test

Testing of the installation was conducted throughout the development process. In the early stages more primitive methods such as using a web browser to see the output of the servlet were used. As the installation became more operative, tests with actual audience could be performed. These tests were filmed, and the audience was interviewed before, during and after the sessions.

User tests and interviews

At the end of the Tapet development process, we had the installation tested by several test subjects outside of the team that had been working on it. The participants included both people from the Intermedia research centre, as well as people with no direct relation to the project. Talks and in-depth interviews with these test participants granted valuable knowledge used not only to answer the research questions of this thesis, but to fine tune the Tapet system.

There were a total of seven test subjects who were interviewed, chosen to fit the role of audience who know of digital art and interactive art in general, but with no direct knowledge of this particular system.

Video capture and analysis

Most of the user tests and interviews were video-taped, preserving the sessions for more detailed analysis at a later stage. These proved valuable in comparing the effects of the different patterns of interaction. In total, about 125 minutes of video was captured during these sessions. By using the data collected through the interviews, observing the user tests and analyzing the video material, this thesis has contributed the identification and specification of five System Response Patterns based on their linearity and homogeneity of response to user interaction.

Reflection

The reflection stage of the learning cycle was to some degree contained in meetings, where multi-discipline discussions would cover the different aspects of what had been experienced in previous use/test sessions. As possible changes and details were laid out, a new iteration could begin with a new design component.

5 Case description: “Tapet”

Acting as base research for the theoretical aspect of this thesis was the actual design, implementation and deployment of an installation of digital, interactive art. Its basic idea is that of mediating dance in a non-traditional manner. The audience enters an area, preferably an enclosed room through a single door. On the opposing wall is a canvas, on which an image of a dancer performing several forms of motion/dance in front of a tapestry is projected by a computer. The tapestry is that of a forest, identical to the tapestry decorating the wall behind the audience. Depending on the reaction-pattern used, the dancer will at one point turn her back to the audience while still dancing. Simultaneously, the static image of tapestry behind her will be replaced by a delayed video stream of the audience. The audience will now be watching themselves observe the dancer from the opposite direction.

The installation also includes two cameras, discretely positioned to avoid unnecessary disruption of the scene. One, a simple web-cam, is used to track the motion and position of the audience. The second is a better, more sophisticated camera which is used to film the audience in front of the tapestry. At given intervals depending on the interaction pattern used, the static image of the tapestry behind the dancer will be replaced by the video recorded of the audience. “Tapet” is Norwegian for wallpaper, hence the name of the project; Tapet.

Figure 11 Idunn Sem and the author; “How do we attach the wallpaper to the wall?”

The following subchapters will give a description of the entire Tapet system.

The first two parts cover the external factors; the actual physical set of the installation as seen from without and a description of the video material, its use and change throughout the development process.

The next three sections describe the inner structures of the Tapet client and server components, as well as the business logic contained in the server.

Following are two sections describing the major ‘inventive’ solutions and adaptations that were found and used, and the most significant optimizations performed.

Finally the results from the interviews and video-analyses are presented in the form of the audiences’ perception of the interactivity in Tapet.

5.1 The set

Physically, the Tapet installation consists of two cameras, a projector with corresponding canvas, two speakers, two Macintosh computers, two Windows computers and a patch of artificial grass indicating where we want the audience to stand.

Each computer has a specific task in the installation. The most basic is the first Mac, which does nothing other than play ambient background sounds at regular intervals, the singing of birds and other forest-like sounds. I call this computer “MAC1”.

The other Mac, “MAC2”, receives a live video feed from the video-camera, encodes it with Quicktime broadcaster and publishes it through the Darwin Streaming Server.

The first of the Windows computers, “PC1”, is the one containing the business logic. It runs the Apache Tomcat webserver, allowing the Java servlet to be published over HTTP. Whenever the servlet is requested, it will check what input has been received from the webcam since last update, calculate the next video clip and return only its name as http content. The final computer is the video client. It will request the name of the next clip to play from PC1, retrieve this clip from its own instance of Apache Tomcat and, depending on the clip that is played, key in either the static image of the tapestry or the live feed obtained from MAC2, as background.

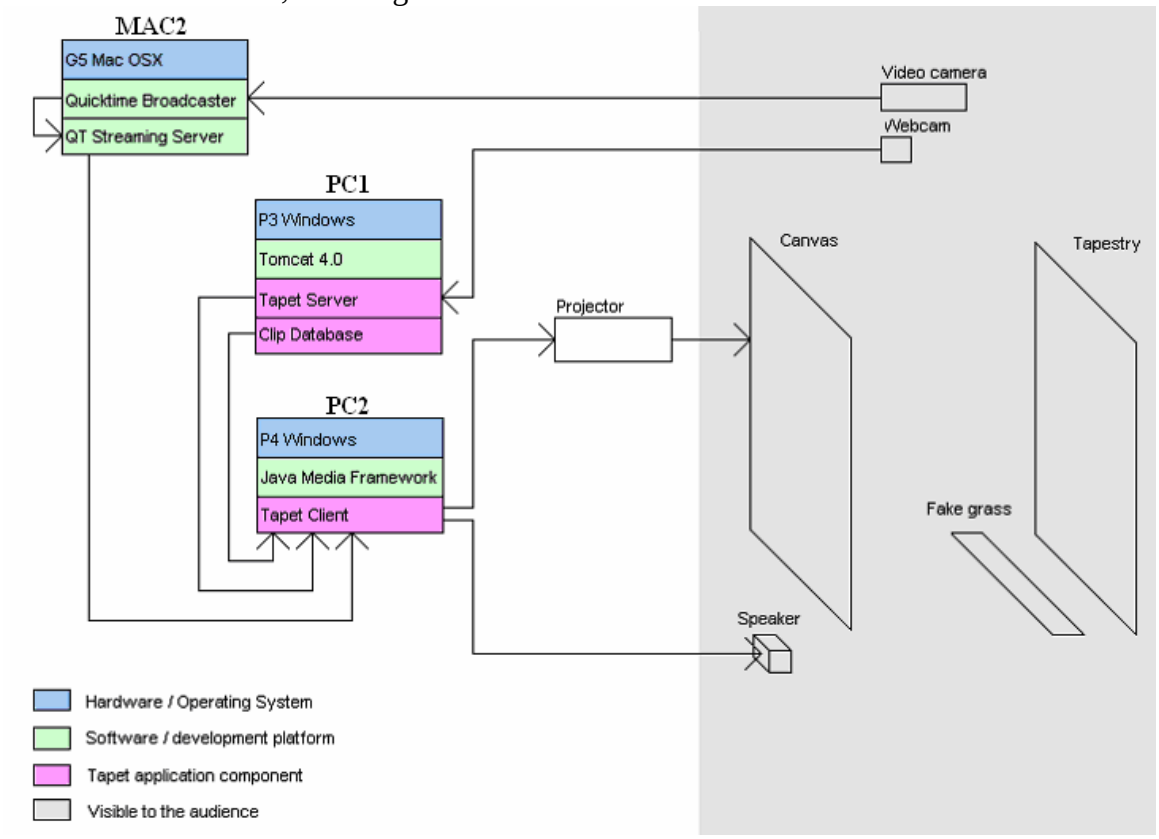

Figure 12 Tapet systems architecture before the introduction of ambient sounds

As shown in figure 12, there are two main computers involved in running the Tapet software; a server and a client. While the base architecture of the application is built along the lines of the Model View Controller principle, the choice of HTTP as control protocol gives a server/client aspect.

The server component is a servlet acting as the controller component of the MVC pattern, instantiating a model component which contains a state machine representing the interaction logic. The client is essentially the view component, using a HTTP_Get method to communicate with the controller.

In figure 13 all the computers running the Tapet installation are shown. The leftmost computer runs the client software (PC1), in the middle are the two Macs. The uppermost Mac is streaming video to the client (MAC2), the one below is playing ambient sounds (MAC1). The rightmost computer is running the servlet that exposes the state machine over HTTP (PC1). Theoretically, one high-end computer could run all the software components simultaneously.

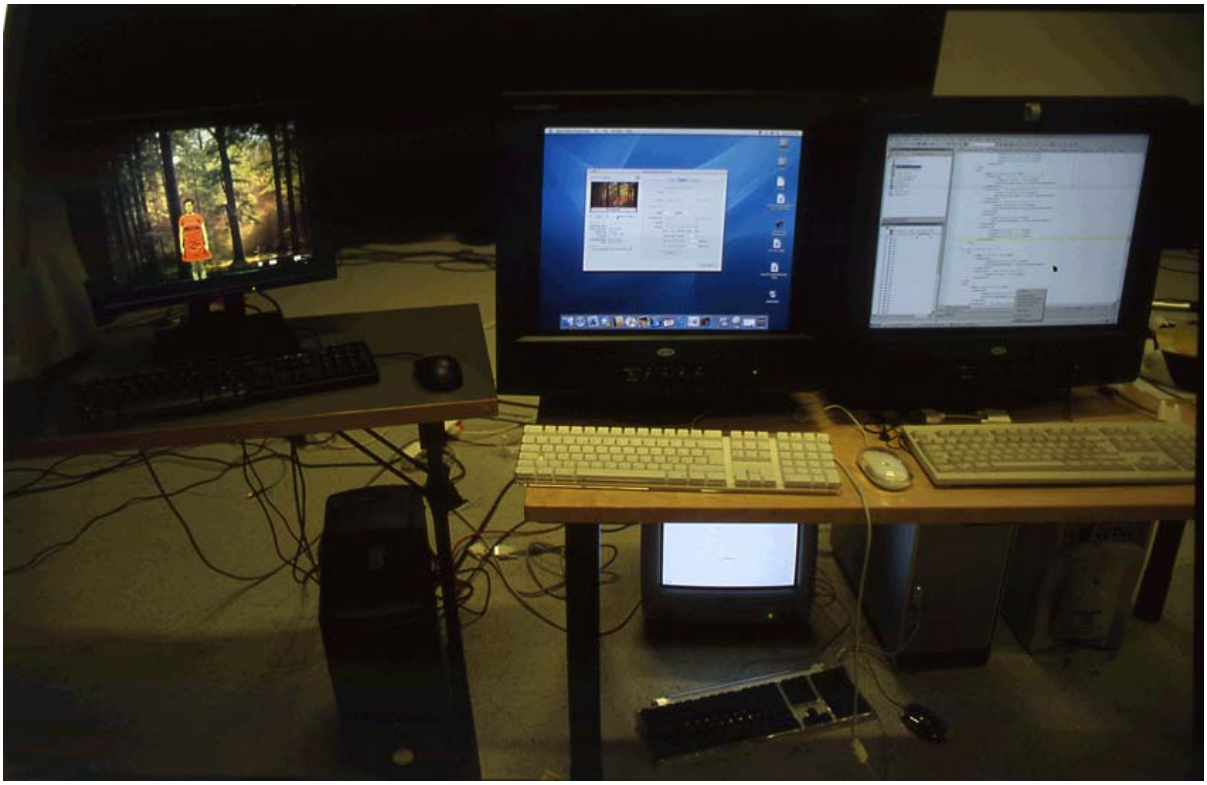

Figure 13 Tapet computer hardware

5.2 Video material – the dance

The actual dance performance is stored as a set of some 40 video files, each covering from nine to 90 seconds of material. The files are stored in the mpeg-4 simple format. The dance is carefully choreographed to fit this pattern, such that every possible transition between two clips can go smoothly. That is, the position of the dancer at the end of the starting clip will have to match her position at the start of the ending clip. This is achieved by having her, the dancer, move around four base positions. Numbered one through four from the left, these positions are where she will start and end each bit of dance in a given manner.

The actual recording of this material was performed at a stage prior to my joining the project. Inger Reidun Olsen choreographed and performed the dance. The choreography was based on the idea and facilitated by Idunn Sem, responsible for esthetic design and 'founder' of the project.

Because the dance will need to be shown in front of different backgrounds, the recording was done with a *bluescreen*. This is a well known technique used by the film-industry. The recording is done with the actor or as in this case the dancer, performing an act in front of a uniformly lit, blue background. The actor will not be wearing any blue, and so the image can later be transformed by replacing the blue with whatever background is desired.

The name bluescreen is in some cases misleading, because actually using blue as the background color is not necessarily the only option. Blue is most commonly used because it complements the human skin, and it has been reported that actors prefer working in a blue environment rather than a red or green room. (Bergh and Lalioti 1999) In Tapet, we actually ended up with a green background, as any inconsistencies within edges or similar would fit better with the forest that would be used for background.

This process is called chroma keying, and in Tapet this was a two-leveled process. First, the raw video files were run through a process that would make them easier to key in real-time. This included removing several bits of recording-hardware visible in the picture; spotlights and occasionally curtains can be seen on the edges. Further, ensuring the uniformity of the base color makes checking what is actual background and not in the image less complex.

The clips were also cropped by removing the top 150 or so pixels. The dancer will never use this part, and thus decoding an area known to be of totally uniform color would be wasted CPU time. She was also scaled up by some 20% to have her appear more life-like in size compared to the audience.

Finally the background color was transformed from green to blue. Having a green background color means an occasional pixel getting past the keying-process would be less visible on the forest background.

Figure 14 Raw video material

Figure 15 Pre-processed video material

Figure 14 shows the original, un-edited recorded material. Spotlights and other artifacts are visible, the background color is not very uniform and the top of the image is still part of the clip.

Figure 15 shows the same image after the video pre-processing procedure has been completed. The background has been changed to green, artifacts removed and the image cropped and scaled.

When the pre-processing is done, the rest of the job must be done real-time. The background is scaled in preparation, using the ImageScaler class.

Figure 16 Background video feed

Figure 16 shows an image taken from the live video feed used as background video. Notice the network sockets and power outlets this forest offers its inhabitants.

When an instance of the foreground image as shown in figure 15, the actual video clips of the dance is ready, it will be sent to an ImageKeyer alongside the background-image from figure 16.

Figure 17 Finished image

In figure 17 the chroma keying is done, and the image is ready to be presented on screen for the audience.

5.2 Server architecture

The business logic of Tapet is contained by a servlet. This web-application contains four interconnected classes, and the specification of a state machine in ‘scene.xml’, as illustrated by figure 18.

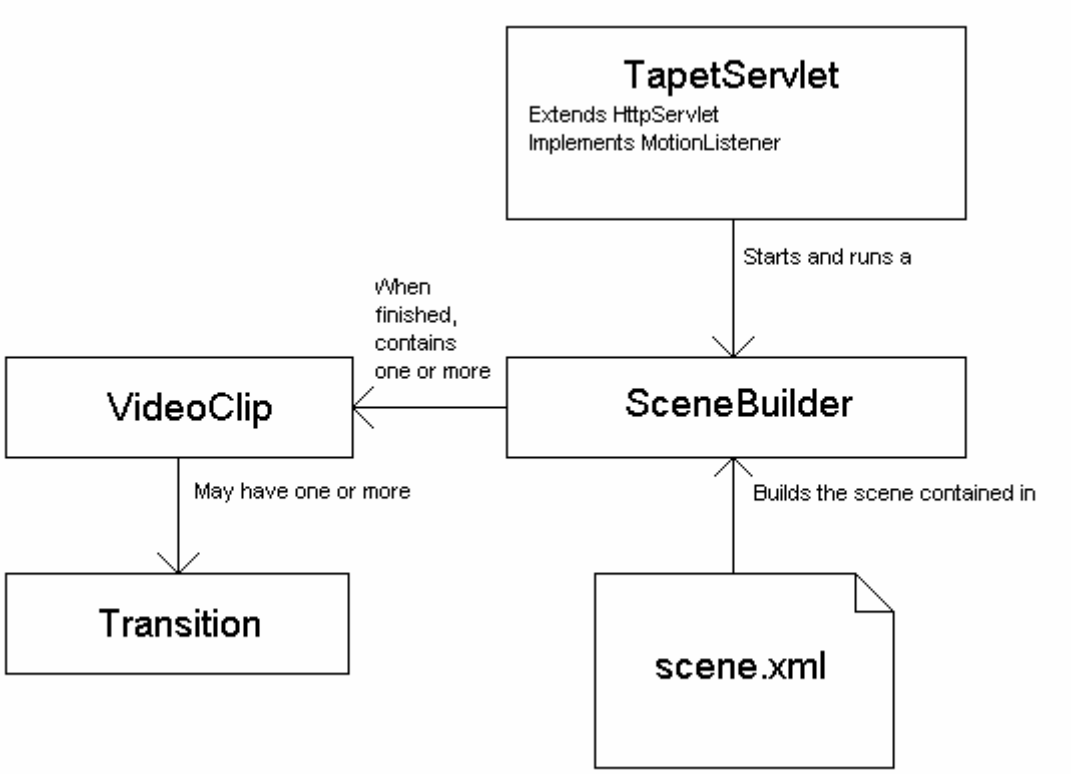

Figure 18 Server architecture

SceneBuilder and TapetServlet are the two main classes of the servlet, both are described in the following sections, along with the xml-file and its format.

TapetServlet

The main class of the application is TapetServlet, which is responsible for answering HTTP requests as well as handling incoming data from the webcam. The actual job of detecting motion in the image is done with the MotionListener class, a part of the Lejos package.

Each image is split into several zones, and motion in either zone will fire an event that TapetServlet will record and use in calculating the next clip. Lejos is the software used in controlling the Lego robotics kits, for motion, sound and video. More information on Lejos can be found at their website; http://lejos.sourceforge.net/.

TapetServlet will initiate the webcam, and start the SceneBuilder with the given xml-file containing the scene logic to be built.

scene.xml

Contained in this file is a representation of the possible sequences of dance based on audience interaction that through the SceneBuilder class forms the specification of a state machine.

The document structure of the xml-file is fairly simple, with <scene> as root element. Each <scene> contains one or more <clip>, which corresponds to the class VideoClip. In turn, each <clip> contains a <name> with the filename of the video clip, and may contain one or more <transition> elements.

Each <transition> element must contain one or more <target> elements, which is the name of the clip the transition points to. Additionally, requirements may be specified that must be fulfilled for the transition to occur. There are three different requirements available;

- <timerequirement> allows a test on the number of seconds that have passed since time was last reset.
- <idlerequirement> allows a test on the number of seconds with no input from the webcam.
- <positionrequirement> allows a test on the position TapetServlet has decided is currently the most valid for the audience.

The tests must be of the form [OPERATOR] [VALUE]. Multiple tests can be separated by the keywords “AND”, “OR”. To avoid confusion with the xml keyword-containers “<” and “>”, “smaller than” is defined as “[“ and “larger than” defined as “]”.

Examples of valid requirements:
<timerequirement>]10 AND [50</timerequirement>
<positionrequirement> =2 </positionrequirement>

Finally, the <clip> element may specify <resettime>, which will reset the timer at the end of the clip in progress. This because the timer is used to measure how long a member of the audience stays in any given position, and thus the time the dancer spends moving from the previous position to the new one should not be counted.

SceneBuilder

SceneBuilder will iterate through the specified xml-document. Every time a clip-element is found, all its transition-children will be instantiated as Transition objects. Finally, a VideoClip object will be created, containing the list of transitions.

On the first run, SceneBuilder will return the first clip in the xml-document. Subsequent calls will have it go through every transition of the previous clip, and return the most suitable one.

5.4 Client architecture

The client part of the Tapet architecture is, as with the server, a set of interdependent classes. It is built on the Java Media Framework(JMF), a part of Java aimed at presentation of media. See chapter three for more information on JMF.

The correlation between the classes in the client is described by figure 19.

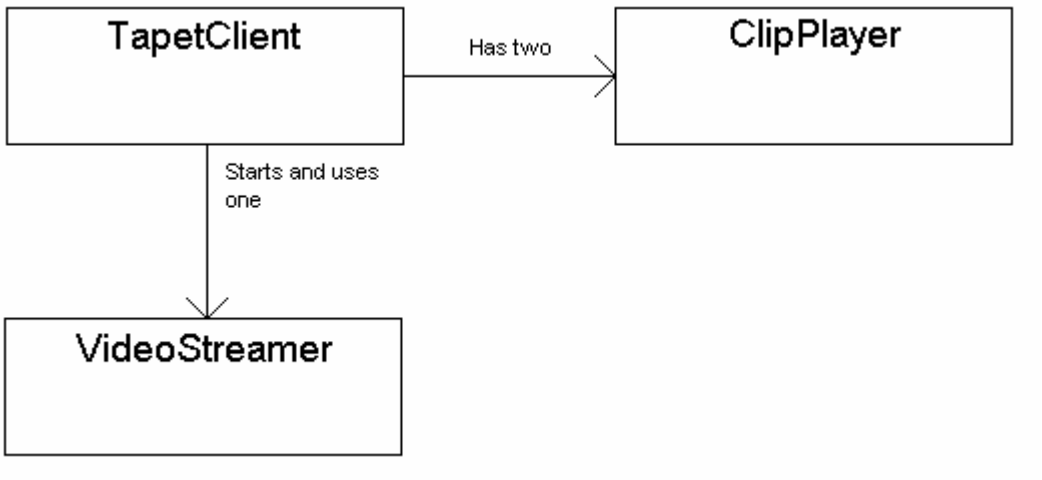

Figure 19 Client architecture

Because several effect-class operations had to be performed on the different video-inputs, a package of tools for this purpose called “RGBImageTools” was developed.

The different classes used in the Tapet client utilize several of these tools. The sections of this chapter describe each of the classes shown on the above illustration, their respective use of the RGBImageTools package, and finally a closer look at the different tools offered by the package.

TapetClient

The main class of the application, TapetClient does not actually contain any JMF code. This is abstracted to the extensions VideoStreamer and ClipPlayer. In the init() phase, the VideoStreamer and ClipPlayer objects are created, full-screen mode is set and the mouse cursor removed. The first video clip to be played is requested from the server via an HTTP request, and one of the ClipPlayer objects is started with this media file or stream.

The most vital method of TapetClient is called displayImage(). When called, this method will receive the foreground image to be displayed. It will collect the most recent update

of the background from VideoStreamer, and use RGBImageKeyer to perform chroma keying between these two images. The result is then displayed on the screen.

VideoStreamer

The VideoStreamer is a class used to receive and prepare the background video of the audience that is constantly broadcasted. At creation it receives the URI of the video; this could be a file or an RTSP-stream.

It runs as an independent thread, offering a public array of 32bit Integer values always containing the most recent image that was decoded from the input-stream. Because the incoming video may be of a different dimension than what is requested, an instance of the RGBImageScaler is used to resize every image before publication.

Additionally, a copy of the first image that was received from the stream is stored for later use, as the background of the application will not always be live video. What to publicize, live video or the stored image, is decided by the variable useVideo. When this variable is changed, a fade-like effect is initiated between the two to avoid a sudden appearance of the audience. This is done with an instance of the RGBImageFader class.

ClipPlayer

As the background video, the running foreground is an independent thread, updating whenever a new image is available from the file it is currently playing. Unlike the background however, the ClipPlayer will call the method displayImage() in the parent TapetClient when it has finished decoding an image. Only when displayImage() has finished, will paint() in TapeClient be called, and the screen updated.

Because the foreground-video, that is the actual video of the dancer, is the one with the highest frame rate, the size of the video actually drawn is decided by the ClipPlayer. This way, the foreground video does not need to pass through a RGBImageScaler instance before operations such as the chroma keying can commence.

RGBImageTools

Initially the chroma keying of the two different inputs; the background with a static image or video of the audience, and the foreground video-clips of the dancer, was to be done in hardware. A special box would accept two sets of input, both of DVI format. The two would then be combined, and a single VGA output signal could be sent to the projector for display.

Upon testing however, it was discovered that the result of the keying was so poor this solution was not viable. A better hardware box was available, but its purchase would be far too costly for this project.

Had the hardware option of chroma keying worked, hardly any of the CPU-intensive image-operations would have been necessary. Obviously, the actual chroma keying would not be needed as it was done in software. Further, no scaling of either input would need to be performed. Both could just be displayed fullscreen, regardless of their respective resolutions. The fading-effect of the background, when the static image changes to video or back, is the only thing that would still need to be done in software.

Unfortunately the hardware solution was not feasible, and all of these operations had to be done in software. To facilitate this, a set of classes were made of the required operations, optimized for performance rather than scalability. These form the package RGBImageTools, which contains the following classes:

- **RGBImageKeyer**
  This class performs chroma keying between two images, on either the red, green or blue color channel. The output image is also color-corrected by values that can be set with the setBalanceValues() method.
  Both of the input images, as well as the output, are 32 bit integer arrays of the ARGB format.

- **RGBImageScaler**
  Specifying the output size of a Player in JMF is possible via the setPreferredSize() method, however this only applies to the final renderer, and does not affect effects, codecs or other plug-ins. The background and foreground videos are not necessarily of the same size, and thus either needs to be adjusted before chroma keying can be performed.

  To improve performance, the class takes input and output sizes at instantiation, and precalculates a distortion-table. Whenever an image needs rescaling, the class creates an output array, iterates through it and fills in the pixel hinted by the distortion-table.

- **RGBImageFader**
  Whenever a change is requested to the background-stream, switching between the static image and live video, a transition phase is initiated to avoid the feeling of choppiness an instant switch may give.

  RGBImageFader performs this phase by doing a linear interpolation between the two over a given number of frames. Currently this is set to 15 images, just over half a second with the frame rate set at 25.

5.5 Business logic

Tapet can be split into two main parts with regards to logic; the background and the foreground. The background can be one of two states; a static image taken from the start of the video feed, or the actual video feed, which is a slightly delayed live video stream of the audience.

The foreground is split into four major positions, labeled one through four. For the dancer these are four different physical locations she can occupy on the canvas, one being the leftmost and four being the rightmost. For the audience the positions follow the same pattern, making it possible to determine if the dancer and the audience are located opposite each other. In each position the dancer can be in either of four modes with differing patterns;

- Passive mode has the dancer move only slightly, while waiting for some input from the audience. Each position has three unique passive clips, the one shown is chosen on random.

- Transition mode is entered when the audience is in a different position than the dancer. She, the dancer, will then move from her current position to where the audience is believed to be. The transitions are also variations of dance, and every position has a transition to every other position. Every clip can move directly to a transition clip, with the exception of position three dance clips. Here, the dancer will have to move back to her starting point in position three first via a move-back clip.

- The actual dance mode is entered when the audience and the dancer are in the same position. There are three different dance clips for each position, and which is shown is chosen randomly with the exception of position three. In position three, the clips must come consecutively, first dance1, followed by dance2 and ended by dance3.

- After a set time or number of clips has been played consecutively without the audience changing position, the dancer will turn her back to the audience while still dancing. There is only one clip for each position in the turned mode. This is also the flag for switching the background to the video feed.

- Additionally, a set of four clips define the overstay mode. These are actually just copies of the passive clips, and are used when the audience has remained in one position for too long. The only difference between the passive mode and the overstay mode lies in the possible transition targets. Passive clips will be exited if motion is detected in any position, while the overstay clips ignores movement in the current position of the dancer.

The business logic of Tapet is contained by a SceneBuilder object, realized by parsing and interpreting the accompanying xml document. The advantages of keeping the

business logic of any application as a separate class like this are easier maintenance and modification of code, better readability and higher extendibility to name some.

Keeping the base of this state machine in an easily modifiable xml-file furthers these advantages. Most notably, changing the behavior of the application can be done with any text-editor, with no knowledge of the actual layout of code, classes or even programming at all.

In Tapet this has proven very useful throughout the project's life cycle. During the early stages of development, the different backgrounds of team members made communicating ideas and writing specifications difficult. While the basic ideas were understood, some misinterpretation of the behavioral pattern of the system occurred. Additionally, several changes to the intended system were conceived throughout the development and testing. Once detected, fixing these was almost exclusively a matter of editing the representation of the state machine.

After the first set of tests using audience, similar changes were made necessary. From the response of the test participants, we formed four different setups in the form of individual xml-files whose characteristics I will discuss in detail in the next chapter.

5.6 "Hacks" and ad-hoc solutions

During the process of implementing the presentation layer, several issues would surface that required non-traditional solutions. Some of these solutions had to be put in software or done in modules where they ideally did not belong, often due to restrictions in time, funds and available hardware. Certain quirks in the Java Media Framework also required sorting out.

While using Java for the business logic and communication layers offered little or no problems, the presentation layer posed a challenge. Traditionally Java is thought of as slow and clumsy when it comes to graphics. What follows is a description of the most notable of these problems, and the adaptations and solutions that had to be found to ensure quality performance from the presentation layer.

The problems and their solutions have been categorized and are described in their respective subchapters as follows:

- Hardware related problems
- Java related problems
- Making fullscreen fill the screen
- Transition between live and static background
- Achieving seamless transitions between video clips
- Chroma keying

Hardware related problems

Other than the components originally planned such as the chroma keyer and the delay-box, the main source of hardware headache were the cameras. Finding a suitable location for the webcam was a challenge. In a multi-purpose studio with many simultaneous applications, the need for an easy way of calibrating the different zones of motion-detection arose. A small application displaying the image produced by the webcam, as well as an outline of the different zones made this job a lot easier.

Further, the video camera had a few quirks. Adjusting the white balance of the recorded image was possible, but saving the settings on the camera did not work. Thus the only available settings were two pre-defined ones, and neither was well suited to the special lighting required on the set. To compensate for this lack of adjustability, an extra layer of transformation was added to the chroma keying software, allowing for independent scaling of the red, green and blue color layers of both foreground and background video channels.

There was also the case of a gray line on the bottom of the video stream coming from this camera. To rule out the possibility of this being a problem with Java Media Framework, we tried reading the stream with the Quicktime player. The same grey stripe appeared here. In an effort to find out where this stripe originated, the Darwin Streaming Server was omitted, and several broadcasters other than Quicktime broadcaster were tested – all to no avail. Finally we had to conclude that the gray line came from the camera itself.

With no substitute camera available, the line is instead masked out in software. The class responsible for handling this is the ImageScaler. It can be given a number of pixels to ignore at either side of the incoming image when performing the rescaling. Because the scaling is done with a distortion-map, the actual masking does not constitute any runtime overhead whatsoever.

Java related problems

The most conspicuous limitation of Java Media Framework, possibly next to the performance issues general to all Java, is that of receiving video streams. The support for decoding different file and compression formats of media files is massive, especially with some of the open source plug-ins that are available. The support for decoding video streams however, is severely limited.

With the first set of performance tests, we find out that having the resolution of the background video stream equal to that of the foreground does not impose much of a performance loss. The rescaling of the smaller image takes what CPU-time was freed. In realizing this, we try increasing the resolution of the image transmitted from Quicktime Broadcaster. What came out was a rescaled image of the original resolution, 352x288. Testing with other clients to make sure this is no fault of ours confirms that the resulting image is indeed of higher resolution, but a rescale of the 352x288 version. Browsing through Apple’s Quicktime Broadcaster forums reveal that in fact, higher resolutions will

always be rescaled versions of the 352x288 image, regardless of codec used. (QTBLists 2004)

Finding alternative broadcasters is a complicated matter, due to JMF's limited video stream decoding ability. Not only must the stream be encoded in a way JMF can decode, but the packet must also be of a recognized format. Figure 20 shows the available compression codecs and packet wrappings JMF is able to read, along with any resolution restrictions.

Compression codec	Encapsulation	Maximum resolution
H262	H262	352x288
H263	H263	352x288
Jpeg	Raw, Jpeg	N/A
Mpeg1	Mpeg1	N/A
Raw	Raw, Jpeg	N/A

Figure 20 JMF broadcast codecs

As the table above shows, there are four different compression technologies that can be used when transmitting video as well as sending the data in its non-compressed (raw) form.

Several alternative broadcasters were found and tested. Sorenson Squeeze and Mpegable Broadcaster were both able to transmit video at any resolution, but only using their own versions of mpeg-4 encoding. With JMF not being able to read any form of mpeg-4 video stream at all, this was of no use.

The open source solution VideoLAN proved to be the most likely alternative to Quicktime Broadcaster. Supporting even more ways of encoding than Quicktime, transcoding between formats and multiple outputs this is an excellent program. The one flaw it has is the encapsulation, which is limited to MPEG-TS. While the most common way of encapsulating mpeg-video streams, it is not recognized by JMF and thus even VideoLAN had to be dropped.

The end result is a background video running at 352x288, scaled up to the resolution of the foreground. While not entirely desirable, the fuzziness of the projected image to some extent compensates this.

Some problems concerned the foreground rather than the background. When a clip is being prepared to start playback, its Processor goes through several phases as explained in chapter three. Before the Processor reaches the "prefetched" state, the first few frames of the clip are sent to the frame handler as ready. The clip can not start playing before it reaches its "prefetched" state however, and additionally the caching process may be finished long prior to the previous clip being done. To avoid loosing these first few frames, they are cached in an internal image-array, and as soon as the clip is started they will be sequentially fed to the frame handler.

The last frame of a video track behaves in a slightly odd way as well. Possibly to accommodate the needs of a media player-like application, the first frame of a clip is repeated after the last frame. In a video containing 318 frames, they will be displayed in the following order; 1, 2, 3 ……. 318, 1. The processor will send additional information alongside each frame to the frame handler, and among this information is a sequence number that can be extracted with the getSequenceNumber() method. The number of the first frame that appears after the last one is equal to that of the last frame, giving sequence numbers; 1, 2, 3 …… 318, 318. The problem is easily omitted by ignoring a frame if it has the same sequence number as the previous frame.

A side effect of the additional frame is a slightly skewed calculation of framerate. The JMF Processor class has a built in synchronization feature that should take care of problems such as this, but due to the first few frames being fed to the handler before the clip is ready to start playing the synchronization fails to evaluate the first frames properly.

This resulted in a need to write a custom synchronization module. The synchronization is quite simple, and located in the TapetClient class. The total number of frames in a clip, subtracting the extra last frame, is divided by the framerate to get the number of milliseconds that should pass between each update of the screen. If this number of milliseconds has not passed since the last update when paint() is called, sleep() is issued with the appropriate amount of time.

In JMF there exists functionality to turn the built-in synchronization module off. However, when using custom synchronization the EndOfMediaEvent interrupt is not triggered at the end of a video clip. This is not due to synchronization happening either faster or slower, EndOfMediaEvent is not triggered at all. A Processor is not de-allocated properly unless it has gone through its EndOfMediaEvent, and can not be re-used until it has been de-allocated. In order to solve this problem both the original, built-in as well as the custom synchronization modules are used.

The idea of switching to full-screen mode in Java was also slightly problematic. While functionality for this does exist and the actual switching does not pose any problems, having the 720x576 rescaled to fill the entire full-screen window proved to be a considerable addition to the CPU-load of the client application. The paint() method takes the screenBuffer object, which is an array of 32-bit Integer values, and creates an Image object from it. This Image is subsequently drawn onto the center of the canvas with the following code:

```
Image drawThis = createImage(new
MemoryImageSource(videoWidth,
     videoHeight, screenBuffer, 0, videoWidth));
g.drawImage(drawThis,leftMargin, topMargin, null);
```

There are basically two ways of rescaling the image to span the entire screen; either before the Image object is created by manipulating the screenBuffer Integer array, or by using Java’s built-in method for rescaling Image objects. Expecting the built-in method to

be considerably faster, along the lines of System.arrayCopy(), the first attempt was done using the getScaledInstance() method of the Image object.

The rescaling can be done using a number of different algorithms, each with different levels of quality and performance. (SUN 2003) In order to maximize performance, the "SCALE_FAST" version was used in the following manner:

```
Image scaled = drawThis.getScaledInstance(screenWidth,
     screenHeight, Image.SCALE_FAST);
```

Unfortunately this method proved very slow, even with the "SCALE_FAST" flag. A loss of framerate would occasionally occur, and so this method had to be abandoned.
Using an instance of the ImageScaler class was faster, but still had an impact on framerate when the highest levels of anti-aliasing and blur were used in the chroma keyer.
In the end we decided on not scaling the end image at all. Using the 800x600 screen mode, there is only a tiny black border surrounding the output image. By using the zoom function on the projector, the border can be pushed outside of the area that is viewable by the audience.

Finally, even in fullscreen mode the mouse cursor is still visible. Failing to find a way of turning it off, a single-pixeled transparent .gif image was used as a substitute for the original cursor effectively removing it from sight.

Making fullscreen fill the screen

While using the zoom function of the projector pushes the border of the image provided by the Tapet client out of the canvas, this approach creates a new problem.

During startup, the application will take the first frame from the background stream and use this as a static image. Because this video-stream is in fact ten seconds delayed, the black border will still not be in the stream when the application starts, even if the screen mode is set before the background-stream is started.

Avoiding the problem could be done by keeping the projector shut off till the application finishes its init() phase. Though possible, it is impractical to do this every time you want the application to start – so a ten second delay was added before taking the static image.

Transition between live and static background

An important part of the Tapet installation is the sequence where the dancer turns around, and the delayed video feed of the audience is projected behind her, allowing the members of the audience to observe themselves observe the dancer from the other side.

Initially the transition from a static background to the video feed was supposed to be instant. Starting with the first frame of the clip where the dancer turns, the background would be the video feed.

With the first few sets of user tests, this was found to be too “choppy”. Several test participants commented on this, thinking something was not working properly.
The only viable option for making the change of background appear gradually was a linear “fade” between the two; static and video feed.

Starting with the first frame of the corresponding video clip, the background will be a mix of the two with a given weight to each. Starting at 1 for the static image and 0 for the video feed, these will gradually switch and end at the opposite as demonstrated by the following code:

```
          if(useVideo) {
            scaleFactor += (double)(1 /
(double)transitFrames);
            if(scaleFactor>1) scaleFactor = 1;
          }
          else {
            scaleFactor -= (double)(1 /
(double)transitFrames);
            if(scaleFactor<0) scaleFactor = 0;
          }

//Scale = 1 means use only the video image
          if(scaleFactor == 1)
            imageArray = scaler.scale(lastImage);
//Scale = 0 means use only static image
          else if(scaleFactor==0)
            imageArray = largeBackgroundImage;
          else
            imageArray =
scaler.scale(transmuter.transmute(lastImage,
smallBackgroundImage, scaleFactor));
```

Achieving seamless transitions between video clips

The very first version of the Tapet client was built on IBM’s Mpeg-4 toolkit, intended only as a means of testing the business logic contained in the servlet. Well documented and at first glance quite powerful, the toolkit lacks the ability to access and manipulate individual frames of video tracks. Further the Player object is not recyclable, meaning that playing two consecutive clips with the same Player is not possible without re-instantiating it. This causes a highly noticeable delay between the end of clip #1, and the start of clip#2.

With the change to Java Media Framework and the abandonment of IBM's Toolkit, a much greater control over the entire playback process is gained; both during caching, frame access during playback and the end of media event.

Making the transition between two video clips totally seamless was still a major problem, one that appeared repeatedly throughout the development process. Many of the underlying causes have been mentioned in the previous chapter; "Java related problems."

The first few frames of clips are thrown out before the caching process is actually done; this gives the appearance that a clip plays too fast, rather than too slow just after a switch of clips. The first frame of a clip is repeated after the very last one. Even at 25 frames per second, this one frame is very noticeable.

Equally troublesome is the fact that the time spent by the caching process is very irregular. Ranging from 50 ms up to 1500 ms, the initial cache time of one second was occasionally too low. This caused a delay between two clips that would only occur very rarely.

Chroma keying

Having the chroma keying operation give the desired output quality, while being fast enough not to hinder the framerate, was possibly the most challenging part of implementing the Tapet software.

The first set of test-clips were of the form shown in figure 14 in chapter five. As a result, the first keying algorithm had to remove not only entirely blue pixels with RGB-value #0000ff, but anything resembling blue as well as the wallpaper on the left side, and the lighting hardware from the top.

The left and top part were just ignored by starting the keying from under and to the right of those areas. Masking out the blue however, posed a problem by simply checking the RGB-values. By calculating the Hue, Saturation and Intensity of each pixel, an area of color could easily be defined as what should be replaced. This proved to be an expensive solution in terms of CPU-power, and several optimizations had to be made to provide the desired framerate.

The final video clips however, had the blue evened out and changed to a uniform green. Additionally the left and top areas had been clipped, leaving only the part of the video that was actually in use by the dancer. In these new media files the green could be isolated as pure green; #00ff00. Simply replacing this color did not give a desired effect. The edges of the non-green area are still affected by the green, giving a green outline surrounding the dancer.

In figure 21 only pixels of pure green color, #00ff00, have been replaced with the background image.

Figure 21 All-green pixels removed

By using the previous method of identifying the green pixels, calculating the Hue, Saturation and Intensity of each pixel, the border pixels that are “greenish” can be found and removed. To avoid the overhead presented by such a calculation, an easier method lies in simply checking whether the green component of a pixel is higher than both the red and blue component, respectively. (Bergh and Lalioti 1999)

Replacing all the pixels identified with this method ensures that virtually none of the green is left. In figure 22 every pixel where green is the dominant component has been replaced by the background image. All the green is gone, but some pixels are lost in the process.

Figure 22 Dominant green pixels removed

This approach gives very pixelated edges, as well as loosing some of the detail on the outline – especially the hair suffers. The algorithm presented by Bergh and Lalioti (1999) suggests using a linear table of weights to calculate a mix of the foreground and background pixel. The desired effect was not achieved with the video clips in Tapet, but a mix of anti-aliasing and blur solves both the pixelated edges and the pixels that are lost along the outline. A brief explanation of anti-aliasing and blur follows, coupled with their implementation into the Tapet video client.

Anti-aliasing

Anti-aliasing is, as the name suggests, the opposite of aliasing. A line created with a regular Bresenham algorithm is totally aliased, in that it is “mathematically correct”. Only the pixels necessary to form the line are present. To the human eye however, the line will appear choppy and not very pleasing. Anti-aliasing is the process of adding pixels along the edges of the line with different levels of grey, giving a much better looking result.

Figure 23 shows the letter: ‘a’ before and after the anti-aliasing process.

Figure 23 Anti-aliasing

The anti-aliasing of Tapet is similar, but not identical to the process described above. If a pixel has green as the dominant component, but is not entirely green, it will be identified as a border pixel. For each of these border pixels, a value called greenDominance will be calculated. This is, as the name suggests, a numeric value describing how dominant the green component of the pixel is.

This value is used as a weight between the foreground and background pixel at the given location. The more dominant green is, the less important that color is thus more of the background will shine through. This could easily be used on the entire image as the greenDominance of totally green pixels would give the background 100% weight. The performance of the algorithm would be lessened however, thus only the border pixels calculate and use the greenDominance. In figure 24, the use of anti-aliasing makes the edges less pixelated and no color information is lost.

Figure 24 Chroma keyed with anti-aliasing

Blur

Blurring can be described as smearing the colors of a given area into each other, letting them affect their surrounding area. Some detail will be lost, but sharp color variations will be toned down.

The blur used by the ImageKeyer class in Tapet has several levels of intensity. The offsets of every border-pixel are saved as they are found, and when the chroma keying is done these are the pixels that are blurred.

The weakest form of blur takes only these pixels, and gives them the average of their original value, plus the eight pixels immediately surrounding them. The next level of blur will perform the same operation on each of the aforementioned surrounding pixels. The final level blurs every pixel that is not considered entirely green in the foreground image. The overhead created by this final level of blurring is so considerable, that an additional option was added using only four of the surrounding pixels as well as the original color when calculating the average was added.

In figure 25 every pixel that has green as dominant color along with their surrounding pixels are blurred. In figure 26 every pixel of the foreground used is blurred, using only the four closest neighbors in calculating the average to avoid a drop in framerate.

Finally the two effects, anti-aliasing and blur, are added.

With the different variations of blur available, several results are available. While a higher level of blur has a better effect on the removal of edges and the feeling of choppy edges it also causes some loss of detail.

Figure 27 and figure 28 illustrate anti-aliasing combined with blur level two and three, respectively.

Figure 25 Blur on border-pixels

Figure 26 Blur on all foreground pixels

Figure 27 Anti-aliasing with blur on border-pixels

Figure 28 Anti-aliasing with blur on all foreground pixels

5.7 Performance and optimizations

With the immense increase in computing power over the last years, very often the cheaper option when a program is performing poorly is a hardware upgrade rather than the extra hours of code optimization that would be required.

In the Tapet project this was also true to a certain extent; a new computer was purchased partially for the purpose of running the Tapet client. The computers previously in use had only 256MB of RAM, and a fairly slow P3 processor.

Despite this investment, a lot of time still had to be spent optimizing algorithms and code in an effort to improve performance. With the addition of chroma keying to the list of tasks to be performed by the client, this computer would now have to perform all of the following in realtime, running at least 20 frames per second in the 800x600 resolution:

- Receive and decode an H263 video stream for the background video
- Rescale each of the images from the video stream to fit the foreground video size
- Read and decode an mpeg-4 file over HTTP for the foreground video
- Perform a linear “fade” between the static background image, and the delayed background video stream
- Create an output image by performing a chroma keying process on the foreground and background
- Run this output image through anti-aliasing and a blur effect
- Perform playback of the audio track corresponding to the current foreground video, as well as display the actual finished video track

During profiling, seven areas were identified as the major consumers of CPU power: chroma keying, scaling of images, crossfading of images, switching between image formats, updating the non-used parts of the canvas, and updating the used parts of the canvas. Two additional means of improving performance were looked into: bytecode precompilation or Just In Time compiling and the Java Native Interface. These are all examined in the following sections of this chapter.

Bytecode precompilation and Just In Time compilation

By having the bytecodes produced by the Java compiler further compiled into a directly executable program, the overhead of translating the bytecode at runtime can be avoided. Precompilation does have the disadvantage of loosing the platform independence that Java normally boasts. Upon further research however, it was discovered that none of the methods available for precompilation support the Java Media Framework and the idea was abandoned.

Similar to precompilation is a technique known as Just In Time(JIT) compilation, which will compile parts Java bytecode program at runtime. Sun provides a JIT-compiler known as HotSpot, which identifies the most used areas of a program for precompilation. The use of HotSpot was tested, but did not provide measurable performance improvements,

possibly due to the native libraries used in the Windows performance pack version of JMF.

Java Native Interface

The second option that was looked into was the Java Native Interface (JNI) that Sun provides. Using the native API allows for calls to methods in external libraries, for instance a C++ library. The idea was to make the most CPU-intensive methods in C or even assembly language, and call these through the native API.

Again the platform independence would be lost, but if the chroma keying and scaling of arrays could be done as assembly language routines the speed gain would be immense. The use of the JNI fell with the fact that java arrays of any given datatype are not actually stored as the equivalent in the computer memory. An array of 32-bit integer values, for instance, will be stored as an array of 32-bit integer *objects*. In order to access a Java variable from native code, the JVM will first make a copy of the array in a separate, non-movable part of the allocated memory. The native code can retrieve a pointer to this, as well as the length of the array through specific JNI method calls. When the native code has finished using the array, another JNI call must be issued to copy any changes to the Java array and deallocate the native copy. (SUN 2002)

Between these extra operations much of the performance gain is lost. It was decided that the extra work of re-implementing a large portion of code in assembler or C was too much work compared to the gain it might offer.

Chroma keying

The chroma keying process was the potentially most CPU-demanding operation of the Tapet client. For every pixel the red R, green G and blue B components must be extracted and depending on the method used, the hue, saturation and intensity might need to be calculated based on the RGB components of the pixel.

Subsequently the pixel must be decided as either “on” or “off”. A pixel that is “off” has a color close enough to the one used as background when recording the original video material to be replaced. A pixel that is “on” does not. Finally, the corresponding pixel from either foreground or background must be put in the resulting output buffer.

With the early editions of the video clips, using the red, green and blue components directly to determine whether a pixel was “on” or “off” was not sufficient. The variation between the separate clips was too big, and the only way to cover them all was by isolating an area of the hue-saturation color wheel.

Unfortunately, calculating the hue value H of a pixel is a complex issue, and involves the calculation of the saturation S and the intensity I as well. The straightforward way of calculating the HSI values from given red, green and blue values involves several square roots, many multiplications and calls to trigonometric functions.

Profiling the application at this point showed the chroma keying as responsible for more than half the CPU-time spent. To further specify what parts of the process that were the most significant consumers of CPU-time a test was run on the different Java functions included. The base test was set up as follows:

```
long startTime = System.CurrentTimeMillisec();
for( int i = iterations; i>0; i--)
     performOperation();
long timeSpent = System.CurrentTimeMillisec() -
startTime;
```

With the new instruction sets of the x86 machines, a mul (multiplication) takes little longer than a regular add (addition) or sub (subtraction). The values reported by the test affirmed this fact. Memory look-ups proved to be fairly close to the basic mathematical functions in terms of performance as well.

Trigonometric calls however, were not. As shown in the results, calculating arccos values in the same setting was considerably slower;

Time spent add: 2794
Time spent sub: 2844
Time spent mul: 3355
Time spent div: 4347
Time spent mem lookup: 3835
Time spent arccos: 74247

A difference-factor of up to 26 seemed totally unreasonable. Granted arccos is a more complex function than an add, but the low performance of the function was seriously hindering a decent framerate.

The input for the arccos function would only span a limited set of numbers, so a look-up table could be created. Running the same test on the look-up table gave the same results as the previous memory lookups, effectively removing the performance-penalty.

The removal of the Java arccos operation helped considerably, but still the chroma keying process was taking far too much CPU-time. Kender's algorithm for calculating the HSI-values goes a long way in fixing this issue. After some modifications the calculation of Hue could be isolated, and the entire function is as follows:

```
if (r > b && g > b)
     hue = PI2 + atanArray[ (int) (550 + (100 * (
(sqrthree * (g - r)) / (g - b + r - b))))];


else if (g > r)
```

```
     hue = PI1 + atanArray[ (int) (550 +  (100 * (
(sqrthree * (b - g)) / (b - r + g - r))))];
else if (b > g)
     hue = PI3 + atanArray[ (int) (550 +  (100 * (
(sqrthree * (r - b)) / (r - g + b - g))))];
```

The variables PI1, PI2, PI3 and sqrthree are precalculated constants. With this method of calculating the hue, hardly any difference could be seen in the profiler when checking the RGB-keyer against the HSI-keyer.

The necessary framerate had now been achieved with the early clips. However, with the arrival of the final clips that were of a higher resolution, the chroma keying process was again taking too much of available CPU-time.

The solution came with the realization that with a framerate of 25, our dancer can only move so much from one frame to the next. The topmost, leftmost and rightmost pixels that are decided as “on” are stored during the process. When the next set of images arrives for keying, the rectangle within these three pixels and the bottom of the picture is given a 15% buffer on all sides. The chroma keying is then only performed on this rectangle as shown in figure 29, and the remaining parts of the image are copied directly from the background picture to the output buffer with the System.ArrayCopy() method.

Figure 29 Chroma keying area

Scaling

Because the dimensions of the video-stream input that is used as a background may differ from that of the video-clips acting as foreground, one of them had to be scaled to fit the other. As the background-video runs at a lower framerate than the foreground, the scaling occurs only here.

The scaling function Java provides, Image.getScaledInstance(), proved rather slow. Even when using the “SCALE_FAST” option, the results were too slow, in part because of the extra overhead needed to convert between ARGB and Java’s own Image format. A custom scaler was made, performing a linear interpolation on the input image. While better, the scaling process was still too expensive in terms of CPU-time spent.

The solution lay in precalculating the offsets in the source image that corresponded to each pixel in the destination image, thus eliminating the need for any calculations in the inner loop of the scaling.

The downside of the precalculated offsets lies in the fact that only images of the same size can be scaled with the same instance of the scaler. However the dimensions of the video-stream can not change run-time, so in Tapet this was no issue.

Prehandling of video clips

To ease the chroma keying process, several adjustments were made to the original video clips. As mentioned in chapter 5.2, all disturbing elements of the background were removed and the background color changed from blue to green.

In addition to this, we removed the top 150 pixels of all the video clips. Profiling showed a reduction of CPU-load spent decoding the mpeg-4 files by approximately 35%, as well as a gain in free memory. The Tapet client of course, compensates for the clips being shorter by displacing them appropriately.

Image format

A problem that occurs when working with digital cameras, various compression techniques and digital broadcasters is that of color depth, and how the image is stored bitwise.

The mpeg-4 compressed video-files output a 24-bit image, stored in three long consecutive arrays of bytes. The first array contains the red component of each pixel, the second contains the green component, and the third the blue component.

The output of the video-camera filming the audience is a 16-bit image, the five least significant bits detail the blue component, the next five the green, and the five next to the most significant bit detail the red component. Other video formats tested output again different formats, ranging from 8-bit color palettes to full 32-bit alpha|red|green|blue format.

The latter is the format required in order to send the finished image to the screen as an integer array. Performing the transmutations of both the foreground and the background for each image is an extremely expensive process in terms of processing power. Fortunately it is possible to specify the desired output format from the Processor class in JMF. Any necessary conversions will then be done during the decoding process, giving no noticeable performance hit.

Choosing video mode

The Tapet client application runs in full-screen mode. To avoid having to rescale every output image to that of the video display, a video mode is automatically selected that is as close as possibly to that of the resulting output video of the application. Currently the output image is 720*576, and the video mode selected 800*600. A small black border surrounding the output image exists, but is offset by scaling the image slightly on the projector.

Updating the screen

The paint() method of the Tapet client takes the integer array containing the most recent output image of the chroma keying process, and creates an Image object out of it using the createImage(MemoryImageSource) method. The Image object is then painted on the application canvas. If the application is set to debug mode some information is written on top of the image before the method returns.

Occasionally Java will issue calls to the paint method on its own, possibly due to resize events or similar. The array containing the most recent image created might still not be updated since the last time paint() was issued, so the boolean variable “newImageReady” keeps track of whether or not paint is allowed to execute. If not, paint will return immediately.

5.8 Design and Implementation process

Due to limited availability of resources such as staff and funding, non-traditional solutions and processes had to be used in order to reach the end in a fashionable manner. A lot of improvisations had to be made. Technologies were used in ways they were not originally meant to, solutions found crossing borders between different disciplines.

This chapter will cover these improvisations and discuss their relevance to the different software layers mentioned in the research questions, based on the experience and findings gained throughout the Tapet project.

The (adaptive) design process

The plurability of backgrounds of the people involved in creating Tapet was a blessing in the way that it opened for finding solutions in several disciplines. However, this also caused a lot of problems in communicating ideas, assigning tasks across discipline-boundaries and actually getting things done.

A solution that requires intimate knowledge of optics and video-theory can not be done by the artistic designer, just as the implementation of a needed software component is best left to a programmer. With time being a limited resource for pretty much all of the team members, this was a serious issue.

Part of the solution was learning a little bit of everything. With this I mean being able to do the most basic and common tasks of the other disciplines, and understanding the concept of the more complex operations in order to help prepare media material, create meaningful requirement-sets and similar.

This applied to me and the one in charge of esthetic design, Sem, in particular, being the two spending the most time on the project. For example I would show her how to operate the programming environment I used. How to stop and start the application, changing Boolean runtime variables and recompiling in order for her to be able to run tests without me present.

Technical design

A considerable amount of time was spent experimenting, reading up on the different available technologies figuring out which of them would be able to perform the different tasks necessary.

When I first joined the project team, my role was limited to the implementation of the business logic. A technical specification of the system existed to some extent, but not yet on paper. In an effort to sum up how Tapet technically was envisioned, figure 30 was made in an early meeting.

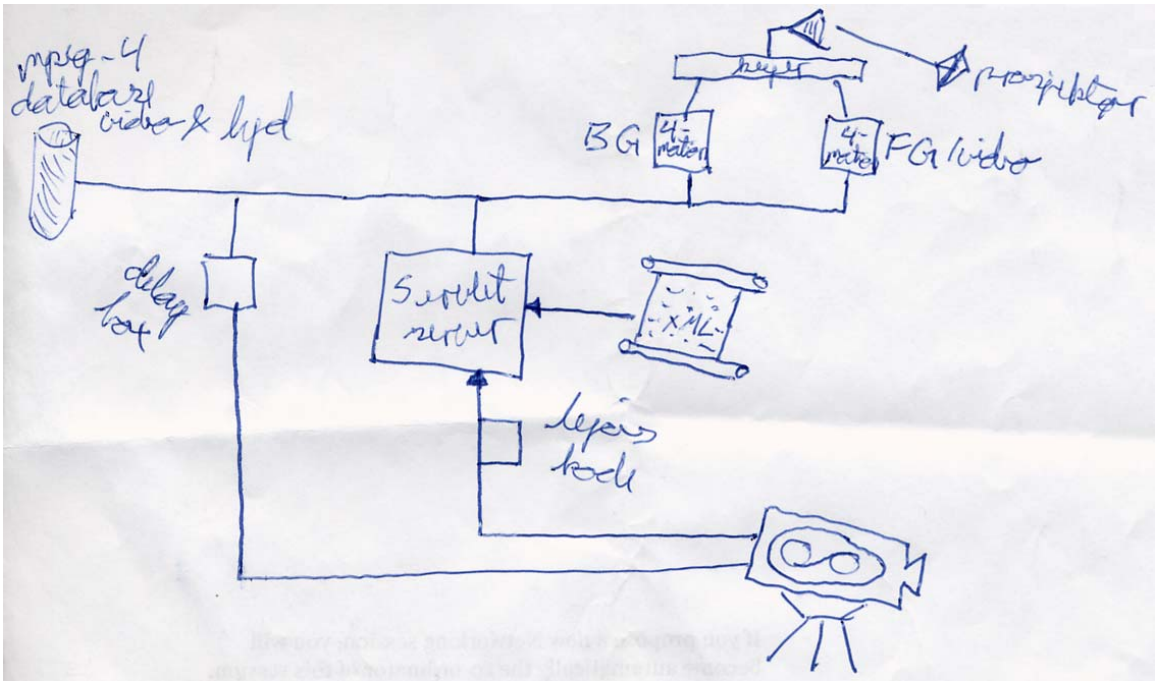

Figure 30 Early technical design

In this setup four software components would have to be developed;

- Lejos code to scan the incoming video data and detect motion.
- The actual business logic, contained in a Java servlet. This would receive the detected motion from the lejos code, and respond to HTTP requests with the filename of the next video clip to be played, or state of background depending on the request.
- The background video client, accepting the delayed video feed and displaying either a static image or the video feed.
- The foreground video client, only performing playback of the video clip the servlet specifies is next.

My responsibilities were at this point limited to the Lejos and the servlet modules. The delay of the video-feed would be done by a special piece of hardware found in broadcasting studios. Similarly, the task of chroma keying the foreground over the background was to be done by another piece of hardware, specially designed for this purpose.

Quickly following, a change was requested where the business logic would be stored in an xml-file as indicated in the specification shown above, independent of the actual Java servlet that would offer it to the rest of the system. There were two reasons for this. Firstly it would considerably ease adjusting the logic, preferably enough to allow the logic to be changed without knowledge of programming. Secondly it could open up for using the system for similar tasks in future installations of similar kind.

Deciding on a format for the xml-based document describing the state machine logic it contained was a fairly straightforward job. The system is basically made up of video clips, that move to a different video clip when finished, based on the variables time and position. Detailed information on the actual tags used can be found in chapter five.

For the actual parsing of the xml document in the servlet there were two main API's designed for this purpose that I considered; Sun's own implementation of DOM as specified on www.w3c.org, and JDOM.

While DOM is a well-known and common standard, it did not seem perfect for the Tapet servlet. DOM requires a considerable amount of memory because the entire document needs to be represented off-disk. More importantly however, is the fact that DOM is designed to be a standard independent of programming language. Because of this it is lacking in terms of behavior expected by typical Java objects. (Lassen)

The open source alternative, JDOM, offers a light-weight and easy to use alternative that proved quite efficient. Only a few lines of code are necessary to do a linear parse of the file which is all that was needed.

The movie clips used as foreground material were originally meant to be put in a database, but were for simplicity put in a folder on the web server hosting the Tapet servlet. The background video fetched from the live-feed of the video camera was streamed through the Darwin Streaming Server (DSS) by Apple's QuickTime broadcaster (QTB). Using this configuration did not come without its quirks; QTB supports most common formats, including those supported by JMF. The encapsulation method used to send the packets containing the encoded data however, are usually limited to QuickTime's own format – which JMF does not currently support. The only format that was found compatible with both QTB and JMF was H263. While this limited the resolution of the stream, QTB has the same limit for every format thus eliminating the difference.

When an early prototype of the servlet with accompanying xml-document was ready some initial testing of the logic could begin. This was done quite simply by moving a pencil in front of the webcam as input, and using a web browser to read the resulting filename. While quite efficient, this method did not graphically show the video clips. To be able to give a more realistic presentation to the rest of the team, I made a simple client using IBM's MPEG-4 Toolkit. Completely disregarding the background and using several seconds to load each clip, this client was far from useable in the actual project, but made both testing and presentation of the system feasible. Figure 31 is a series of three images describing the test-process utilized before the IBM client was finished.

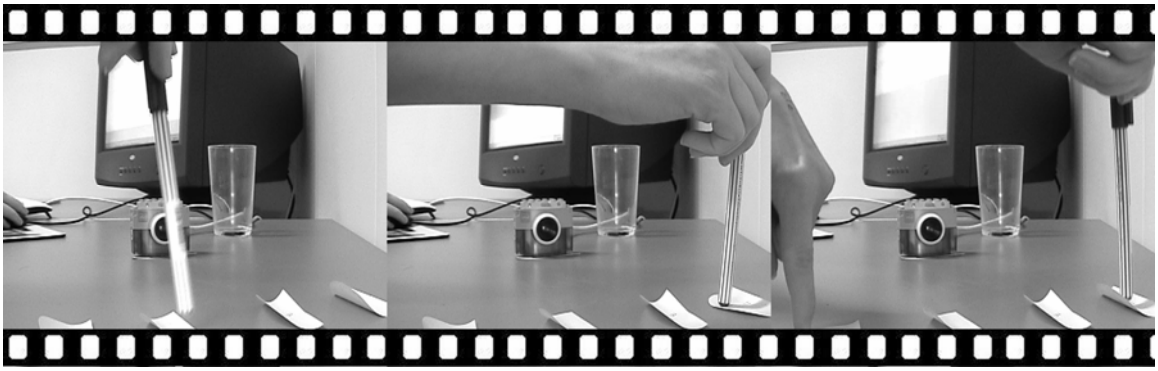

Figure 31 Testing an early version of the state machine.

With the servlet and the business logic up and running, albeit not completely tested, attention was turned to foreground and background clients. To actually see the system in effect these would have to be finished.

The original specification listed Envivio 4Mation as the client software. The idea was to use the Javascript support of 4Mation to create a simple loop, getting a clip name, retrieving the clip from the database and performing playback before repeating. Ideally it would be as simple as the following example;

```
while( true ) {
     String nextClipName = WGet( servletURI );
     mpeg4Media videoClip =
     retreiveFromDatabase(nextClipName);
     performPlayBack( videoClip );
     }
```

What was soon discovered was that the Javascript of 4Mation was not able to read documents over HTTP, nor was there any way of parsing an xml document. With 4Sight ruled out, I was given the task of looking for alternatives.

An attempt at expanding the already existing client that utilized IBM's Mpeg4 Toolkit failed; getting rid of the delay when switching between video clips proved tricky at first, and after a few rounds on the IBM Alphaworks messageboards loosing this delay was proved impossible.

Looking for realistic alternatives in terms of price, time that would have to be spent getting to know the technology and actual time left in the project, three candidates were found:

- Max/MSP with Jitter
  Max/MSP is a programming language with both a graphical interface and Javascript support, allowing easy playback and mixing of music, audio and video. With the Jitter expansion adding considerable 3D support as well as interfaces for input devices, this is a combination used by many to combine and manipulate

audiovisual data. (Cycling74 2005)

The drawbacks of this option is the 850$ price per license, plus the fact that none of the team members had any prior experience with the Max/MSP language. Neither reading data over HTTP or parsing xml is supported natively, but writing plug-in modules in C/C++ is possible.

- Isadora from TroikaTronix
  Isadora is quite similar to Max/MSP and at $350 per license considerably cheaper, but lacks the Javascript support of Max/Msp. As of now the program is only available for Macintosh, with a beta in the works for the Windows platform. Neither reading data over HTTP or parsing xml is supported natively, but writing plug-in modules in C/C++ is possible.

- Sun's Java Media Framework
  The Java Media Framework(JMF) API is quite similar to that of IBM's Mpeg4 Tookit, although far more extensive. Input and output of audiovisual data of a multitude of formats is supported, within the object oriented Java programming language. Feature-fledged as Jitter and Isadora are, they can not match Java in versatility and possibilities.

  The serious drawback to Java is its slow speed due to the virtual machine architecture, requiring on the fly bytecode interpretation. Additionally, JMF has not been updated by Sun for quite some time which forces it to rely on 3rd. party plug-ins for compatibility with newer file and media formats.

Figure 32 illustrates the advantages of the different alternatives. The variables that were the most important to our project are listed; price of a single license of the technology, playback of mpeg-4 data as either a running stream or a media file, native HTTP read, native xml document parsing, custom module creation and programming environment/language the technology is based on.

Alternative	Price	Mpeg-4 playback	Native HTTP read	Native XML parsing	Custom module creation	Programming environment / language
Max/MSP Jitter	850$	Stream and file	No	No	Yes	Graphical / Javascript
Isadora	350$	Stream and file	No	No	Yes	Graphical
IBM Mpeg4 Toolkit	500$	Stream and file	Yes	Yes	Yes	Java
Java Media Framework	Free	File	Yes	Yes	Yes	Java

Figure 32 Video component platform options

With IBM's Toolkit ruled out due to the delay when switching between video clips, there were only three real options left.

Both Isadora and Max/MSP Jitter would require too much time getting into as well as extending to provide support for either HTTP data transfer, or xml-parsing, should the server containing the logic need to be translated to these technologies.

As the actual playback of clips had been previously tested with IBM's toolkit with satisfactory results concerning speed and framerate, the slowness of Java was at this point not yet a problem.

In parallel with the development of the JMF client starting up, the different hardware devices that were to be used in the system were being tested. The chroma keying machine gave extremely poor results, so bad in fact that using it was out of the question. A better machine was found, but priced approximately 60000 Norwegian kroner. Additionally the delay-box that was supposed to add the ten second delay to the live video feed of the audience proved a problem. The device was meant to be available for loan from a television studio, but it did not work out.

Finding a solution to these missing or non-satisfactory hardware devices had to be done quickly, as deadline was soon approaching. Much of the Christmas of 2004 was spent testing the different aspects of JMF, trying to decide whether it was technically possible to do all of these tasks as an integral part of the foreground video client.

In effect, it would mean that part of the Tapet system would have to receive and decode a network video stream, read and decode an mpeg-4 media file, use a form of software chroma keying to combine the two, and finally output them to the display device – all at a decent framerate.

Upon some further research I decided that this was indeed possible, but the problem of speed was now getting to be a real factor. Using 25 frames per second as the desired framerate, I set up a series of performance-tests on my laptop.

Using different resolutions for the background and foreground, changing the way chroma keying was done gave a fair impression of what performance could be expected despite the fact that many of the software modules were far from finished at this point.

Figure 33 shows the framerate achieved with the different methods of chroma keying and resolutions tested.

Foreground resolution	Background resolution	Chroma keying method	Resulting framerate
640x480	320x252	HSI	17.4
640x480	640x480	HSI	17.6
640x480	320x252	RGB	21.3
640x480	640x480	RGB	21.4
640x480	320x252	None	25.1
640x480	640x480	None	25.1

Figure 33 Chroma keying method speed comparison

The tests revealed several points of interest;

- Because of the overhead caused by rescaling, having a background video stream of a lower resolution than the foreground did not indicate any real performance gain.
- Using the hue, saturation and intensity values of each pixel to perform the chroma keying was at this point considerably slower than its RGB counterpart. The HSI version however, gave much better result in terms of quality with the video material at the time.
- Finally and most importantly, it would appear that my laptop was unlikely to give the desired 25 frames per second at a satisfying display resolution.

While the results of the tests did not give the desired framerate, I was now fairly certain it was feasible to have the video client perform all the necessary tasks real-time. A faster computer that would serve as the video client was ordered, a Pentium4 3.2 GHz with one gigabyte of RAM was considered sufficient. Additionally, multiple code optimizations would further speed up both the chroma keying and other parts of the application.

An additional modification to the architecture had to be made to accommodate the new, small multi-purpose video client. Using the camera as input for both motion-detection within the server and delayed video-stream for the client was not immediately possible. Using a different camera for the motion-detection was far easier, and because the level of detail required for this operation is rather low, a regular webcam was sufficient.

The resulting, final system architecture of Tapet is explained and illustrated by figure 12 in chapter five.

Testing and stability

Already in the early stages of the technical design process several tests had to be performed. With a high level of uncertainty on what technologies to use, in particular for the manipulation and display of video, figuring out at an early stage what was actually possible was essential. This subchapter describes the major tests and experiments that were performed before, during and after implementation of Tapet.

With the completion of the first version of the model component of the system, a test was set up for verification. The test context was primitive but effective; mapping out the areas of movement recognition in front of the webcam, moving a pen in and between the areas was used to simulate audience. The resulting change of video clip being played was shown by filename in a web browser. The process was repeated till the output in the web browser was satisfactory. Figure 31 illustrates the process.

With the completion of the first set of tests of the servlet containing the business logic, a small client capable of displaying the very first test-versions of the video clips was implemented using IBM's mpeg4 toolkit. The test client was made for a test-run before the rest of the Tapet team; it had serious flaws such as a three second black screen while loading the next clip to be played.

When the creation of a video client in 4Mation was proven impossible, it was thought that perhaps the test client could be worked upon to act as a final client in a later stage of the development process. Knowing that the video clips would have to be played consecutively in a way that would hide the transition between clips, the transition time available was approximately 1/25 of a second. (With a framerate set at 25 frames per second)

Close examination and testing of the IBM toolkit revealed that this was in no way possible, and the idea was abandoned. Reading that the toolkit was based on the Java Media Framework, we set up a similar test of JMF and found, fortunately, that JMF through the Processor class was able to run two consecutive clips without pause.

Throughout the implementation process of the video client, performance tests were conducted on a regular basis to ensure a proper framerate could be sustained. With the inclusion of CPU-heavy components such as chroma keying these tests would prove essential. The framerate of the video clips was at one point lowered to 20 to ensure stability of playback, but later optimizations of code allowed the framerate to remain 25.

At one stage of the implementation process, a test was run trying to establish whether it was possible to preload all of the videoclips into separate Processor objects. This would allow instant switching between movies without caching. The Tapet system contained too many videoclips; after 17 Processor objects the system would crash with a fatal exception in the native mpeg-4 codec library.

Having established that caching was a necessity, finding the minimum cache-time was made a priority. During the seconds the system uses to cache the next videoclip, no interaction may occur because the next clip has already been selected. To minimize the effect, the system was set up to run and cache all the videoclips a set number N times. The minimum, maximum and average cache-time was recorded. A small buffer was added to the maximum, and thus the final cache-time was found.

With the implementation closing in on done, a test of system stability was conducted. The installation was left running for an hour, 12 hours, 24 hours, a weekend and finally one week. During the one-week run the client (view) component crashed. The debugging leading to the identification of this error was extra strenuous, due to the difficulty in recreating the fault. The problem was finally identified as being part of the blur-component of the chroma keyer, and subsequently fixed.

The final and most comprehensible set of tests were also the last to be conducted. With the software done and the interaction/business logic as finished as we could get it, an audience who did not take part in the development process was used as test-subjects in the installation. Their reactions were to some degree unified, in that the interaction was too hard to understand – or notice at all. Through reviewing their feedback as well as studying video of their experience, the necessary changes to the system were identified.

Areas of application (display locations)

For the actual display of the installation, we had two different arenas envisioned; public spaces and museums/exhibitions.

For public spaces the most obvious type of locations are subway-stations, bus-stations and other public transport stops where people are bound to make several visits. These locations give the audience time to realize how the installation works, as well as plenty of input for the movement-driven controller of the system. Additional public space locations such as display-windows of shops were considered.

The other main area of application is the typical exhibition or in a museum display area. We applied for an entry in “Høstutstillingen 2005”, but were unfortunately not accepted.

Modification; “bendability”

The possibility of re-use is considered an important component of object oriented programming. The “RGBImageTools” library is an example of code developed for Tapet that can be used in other projects. Similarly the entire software system seen as one large object can have re-use applications. This sub-chapter will describe the attributes and parameters that can be changed in the system, giving an idea of possible other application areas.

The foreground video-clips can be replaced by virtually anything, as long as they are within the range of supported formats of JMF with the FOBS plugin (see appendices A and B.)

The background that currently is a wallpaper can be replaced by anything that can be captured by a camera. As the first image of the video-stream is used for the static background image, anything goes as long as an empty space is ensured at system start-up. For instance the background could be a picture from an underwater scene. The video-

clips that are keyed on top of the background could be fish or scuba-divers giving the entire installation an underwater theme.

The red, green and blue channels of color on both foreground and background may be adjusted independently of each other. This is particularly useful if the camera used to tape the audience is delivering a video-stream that is either too dark or too bright to fit the foreground clips.

The business logic describing the succession of video-clips as well as their dependencies lies in the file “scene.xml.” By altering the logic of the state machine contained within this file the narrative structure of the installation can be accurately set.

Future work

The application running Tapet was finished and is working as intended. However there are some parts of the system that could be improved, as well as some interesting elements that could be added to the design. What follows is an explanation of the issues we have identified as components that could be improved.

The delayed video that works as the background is streamed by Apple’s Quicktime Broadcaster (QTB), through the Darwin Streaming Server (DSS). Unfortunately QTB has a maximum resolution of 352x240 pixels, which makes the background considerably less detailed than the foreground which runs at 720x540. A program that could broadcast at a higher resolution and still match the limited formats of JMF would increase the net impression of image quality considerably.

One such broadcaster is the JMStudio that is bundled with Sun’s JMF. Unfortunately, JMF and JMStudio does not support Firewire cameras, which is the only kind we had available. The higher background resolution would cause more processor usage on that part of the system. If the video was broadcast at the same resolution as the foreground however, the background would no longer need to be resized, thus saving considerable CPU time possibly negating the extra overhead of the higher resolution.

The length of the current foreground video-clips range from nine seconds to nearly three minutes. The shorter clips proved far more valuable because they give far greater freedom in designing the narrative structure. The longer clips give much less room for interaction, as any response from the system would have to wait till the clip is done playing. In the final version of the installation, only the short clips are being used. Having the clips re-made to a maximum of approximately 15 seconds each would increase the complexity of the state machine, but be a major improvement to the available interaction-patterns the installation could utilize.

Adding a z-component to the client would also be an interesting feature. It would allow adding static objects such as trees or cars between the audience, the video-clips or the background. Layers of animation could also be added, and drawn according to z-value to ensure proper masking.

5.9 Audience perception of interactivity in Tapet

Through the test of the Tapet installation with audience, the analysis of the video-taped experiences and interviews with the participants, the interactivity pattern of Tapet evolved through several stages. This section describes the perception of the audience on the interactivity in Tapet, and the impact this had on the state machine.

Prior to entering the installation, the test subjects were asked their expectations on what sort of interactivity they would be offered in the installation. Having seen the camera and knowing the installation as interactive art, they all had expectations of being able to somehow 'control' the dancer they had seen on the canvas from the outside. Two subjects also had expectations of being able to participate in the dance themselves; being part of the projected image.

The questions were also repeated during and after the sessions, bundled with more specific questions regarding the experienced effects of effects such as the delayed video and the lack of mirroring on the audience' avatar.

As the tests progressed it became clear that the narrative structure contained in the state machine would have to be altered, and so the questions were repeated on each of the iterations with versions. Each iteration had a different set of participants.

The first test contained the originally planned narrative structure, where the dance would, in each of the four base locations go from an idle clip through one variation of dance, turn and perform another variation of dance with the audience as background, perform one more variation of regular dance and then return to the idle clips.

In this setting the feedback on interaction with the dancer ranged from a frustrating feeling of not being in control at all, to a "somewhat vague control" of where she would go. One test-subject was convinced of her ability to control how the dancer moved her arms.

The audience rarely stayed long enough in one location for the background to switch, revealing that they could become part of the 'action' on the canvas. In one of the few cases where the background did switch, a frustrated test-participant demanded information on how to get back on the canvas. This was clearly a part of the installations narrative structure that caused curiosity and excitement, but the rarity and multilayered path of getting there made it more of an element of frustration than fun.

A second test was prepared. In this run, the first regular dance clip was removed. In practice this meant the dancer would go from an idle clip straight to dancing with the audience in the background. Before returning to the idle mode a regular dance clip was played. In this setting all the test-subjects were able to enter the background of the canvas, becoming a part of the image.

The delay of the video-stream and the lack of mirroring proved an interesting twist on the audience' avatar, something they appreciated. All but one subject experimented

considerably with the on-screen avatar, and considered the two ‘effects’ to be enhancing to their total experience of the installation. The participant who did not enjoy seeing herself on screen explained that she had a considerable dislike to being filmed, photographed and similar, and thus the idea of being on canvas at all was detrimental to her experience of the installation. It might have been different she argued, had she been entirely alone in the installation – without us watching or filming.

When inquiring about the level of interaction experienced between the audience and the dancer, response was now more positive. Still however, frustration occurred concerning how to get back into the projected image and the feeling of interaction after the audience had been removed from the image was low.

One point of interest was the fact that one participant had the impression he could affect the dancer’s ‘mood,’ though he was uncertain as to how. While no such possibility of interaction exists in the installation, it is understandable that the idea can exist as the dancer will show different kinds of feelings in the video-clips through facial expressions.

The third test removed the final regular dance clip, simplifying the narrative structure for each base position to: idle – dance with audience in the background – idle. With this final test the feedback on level of control over the dancer was positive, but the loss of complexity had reduced the longevity of the installation, as well as the number of variations to the performed dance. Still a highlight was getting the audience into the background video-stream, and the effects of delay and removed mirroring were as appreciated as before.

6 Discussion

In this chapter the findings related to the two research questions presented in chapter four are discussed in the subchapters “Conceptual Framework” and “Using Java as implementation platform.” Through the development process this thesis has contributed by defining the concept of System Response Patterns based on the homogeniety and linearity of interaction types, and identified five such patterns as used in the Tapet project.

The actual implementation of the project in Java succeeded. The ramifications of the choice in implementation platform is covered in section 6.2, discussing the advantages Java brings to the more conceptual components of the system: model and Conrtoller. The view component of Tapet is fully functional, but at the cost of many hours spent optimizing program code.

The implementation process started with the Model View Controller design pattern. This proved to be insufficient for the architectural setup, and a new pattern was created. Section 6.3 discusses how the pattern was conceived, its specifications and use.

6.1 Conceptual Framework

The two main factors identified by Lee (2000) as central in establishing a framework for observing or measuring interaction are 1) the setting, and 2) the user’s perception of interactivity in that setting.

Through this chapter Lee’s idea of interactivity will be re-visited, and a new definition aimed at interactive art will be proposed. The setting of the user-tests conducted with the Tapet installation will be identified and the user perception of interactivity before, during and after the experiencing of the installation will be discussed. Finally the interview findings in relation to the level of interactivity will be presented and compared to the components of Lee’s model, finding complexity as a gathering factor for the experience offered by the installation. The results are then used in defining what are denoted ‘System Response Patterns’ (SRP) in this thesis, and identifying the five SRP’s used in Tapet.

Interaction in digital, interactive art

(Lee 2000) defines user intention as vital in his definition of interactivity;

> “*The central idea of the definition is that to be interactive changes should be made in accordance with user's intension.*”

The changes imposed to an installation of digital interactive art will not necessarily be in accordance with the user’s intension. They may be, but the creators of the installation may for instance have set it up to act directly opposite of what they believe audience

intensions to be. Interaction in digital, interactive art must be in the accordance to the intention of the artist or creator, rather than that of the audience.

Reflecting this, I propose a new definition of interactivity in the setting of interactive art:

> "*Interactivity in interactive art is defined as something that changes according to the user's input, in a way that follows the intentions of the artist, based on the input of the audience*."

Here, audience and user are considered synonymous.

Setting

Several forms of interaction may occur between the audience and an installation. Dictated by what form of input sensors the installation includes and reacts on; the different forms of 'user input.' The more common forms of user input devices used in the interactive art setting are covered in chapter 3.

In Tapet the sensory input devices used is a set of two cameras; one detecting motion and the other filming the audience for later use. Communicating with the system thus takes place in the form of embodied interaction.

The response from the system comes in variations in the movement patterns of the dancer, as well as a change in the background scenery. While the dancer is not an avatar of the actual audience, she is rather an avatar for the other participant in the interaction – the avatar of the system. Additionally, the audience is given the video feed of themselves as a personal avatar at certain points in the narrative structure, depending on their actions.

This response can vary in two major ways; time and space.

The variations in space are seen as changes in movement patterns of the two avatars, differing from that of the audience; embodied interaction. The variations in time are for the avatar of the audience restricted to a constant delay of the video-feed, while the avatar of the system has several narrative structures.

Upon entering the actual installation area, the audience all had an idea of what interactive art is and that the piece they were going to witness would be of such a character. The installation was also visible to the audience before they entered the area, as we did not have an entirely enclosed room available. This gave them a certain idea of what to expect, but no further information was offered them.

The installation was shown both to test-subjects one at a time, and to groups of up to three participants.

Interview findings

The installation offers three actual possibilities for interaction; changing the narrative structure of the performed dance, changing the background from a static image to the delayed video-stream, and finally manipulation of one's own avatar by body movements. The only input is by actual movement that is registered by the cameras.

These were mostly recognized by the audience before entering the installation, but the perceived interaction while actually there varied.

We found a connection between the level of expectations and the experienced frustration of not 'getting' the interaction. All the participants of the study had the idea that understanding how the interaction works was essential, but the level of frustration not being able to do so seemed dependant on how much influence they expected to have on the system.

The total perceived interaction was for most the sum of experienced 'control' of the performed dance and the directing of ones own avatar in the image. None of the participants mentioned actually bringing the video-stream of themselves into the background as an option of interactivity.

There were also a few participants who rated the total experience of interactivity high due to the perceived ability they had to affect the mood or direct movement of the dancer. These two perceptions were mentioned in the previous section, and were imagined connections. An interesting note is the fact that they all occurred with the more complex narrative structures – during the two first tests. This might also be related to the level of expectations regarding interactivity. When the expectations of control are not met, they may to some extent have compensated for the systems' lack of response by imagining connections that were not there.

There was a clear correlation between the different narrative structures used in the three tests, and the level of interactivity the subjects experienced with the dance performance. The more complex structures often left the audience perplex as to whether they could influence her (the dancer) at all, causing annoyance and ultimately lowering the total experience offered by the installation. With the increased feeling of interactivity, more specifically the increased understanding of the interactivity the audience reported a higher level of satisfaction.

This could lead to the conclusion that interactivity should be kept as simple and understandable as technically possible. The case of the background avatar however, would indicate the opposite. The delay of the video feed combined with a missing mirror effect (imagine looking in the mirror moving your left arm; and the left arm of your image moves as well) was reported as both fun and interesting.

Reviewing videotapes of the tests later we could see attempts at working out what arm to move, trying to conform their movements to that of the recorded dancer. This deviation

from a 1-1 relationship between audience and avatar was accepted as positive, and was obviously inviting play and experimenting.

Another fact speaking for more complexity of interaction was the few who felt they understood how the system worked in the second test. While it took them some time to 'get it,' the longevity of the installation would increase massively because once they had figured out how to move her, their choices were more varied than that of the third test.

Attributes of complexity

Given the changes in perceived levels of interactivity in the audience, we could gather that the attributes influencing the difficulty in understanding how to work the system, was a combination of homogeneity and linearity, applied to each communication option the installation offers. Here, we define the linearity as the level of similarity between input and output, and homogeneity as the extent of which the same input generates the same output every time.

From the audience interviews and reviewing their actions on tape, these two combined formed the level of complexity connected to each interactivity option, where a lower level of either or both attributes gave a perception of lower interactivity levels in the audience.

As the complexity of the interaction rises, the feeling of being able to affect the installation diminishes, and eventually disappears completely. An image changing at what seems to be totally random intervals and patterns will not be perceived as interactive, even though the changes are based on some input gathered from the audience.

The longevity of appeal also seems related to the complexity of the interaction. While the less complex forms are easily understood and thus instantly gratifying to the audience, they tend to offer a shorter range of options, as well as obviously requiring a shorter amount of time working out the system.

This leads to the conclusion that installations aimed at locations where the audience are likely only to come once, one should choose a type of interactivity that is of low complexity. This suits well in the setting of a museum where audiences come and go, with limited time for each work of art. Exhibitions may also only be available for a short time span.

For settings where repeated exposure to the installation can be expected, a more complex pattern of interactivity could be adopted. Public spaces such as subway stations and movie theatres are places people are bound to visit several times, giving installations with patterns of interaction that are less obvious more time to convey meaning. The realization that the audience has an impact on what is going on may not come at once, giving several layers of possible experience from the same installation.

Figure 34 shows how the two attributes form the total conceived complexity. Most of the test-subjects responded with similar amplitude to each of the attributes, but it could be argued that the interrelation between the linearity and homogeneity is subjective. Figure 34 also displays a line representing where the interaction pattern becomes ‘too easy’, a line representing where it becomes ‘too hard’ and is color coded to represent the type of presentation area best suited to the pattern; public space or exhibition.

The polarity of the axes is reversed, meaning that total linearity and homogeneity is achieved at the origin.

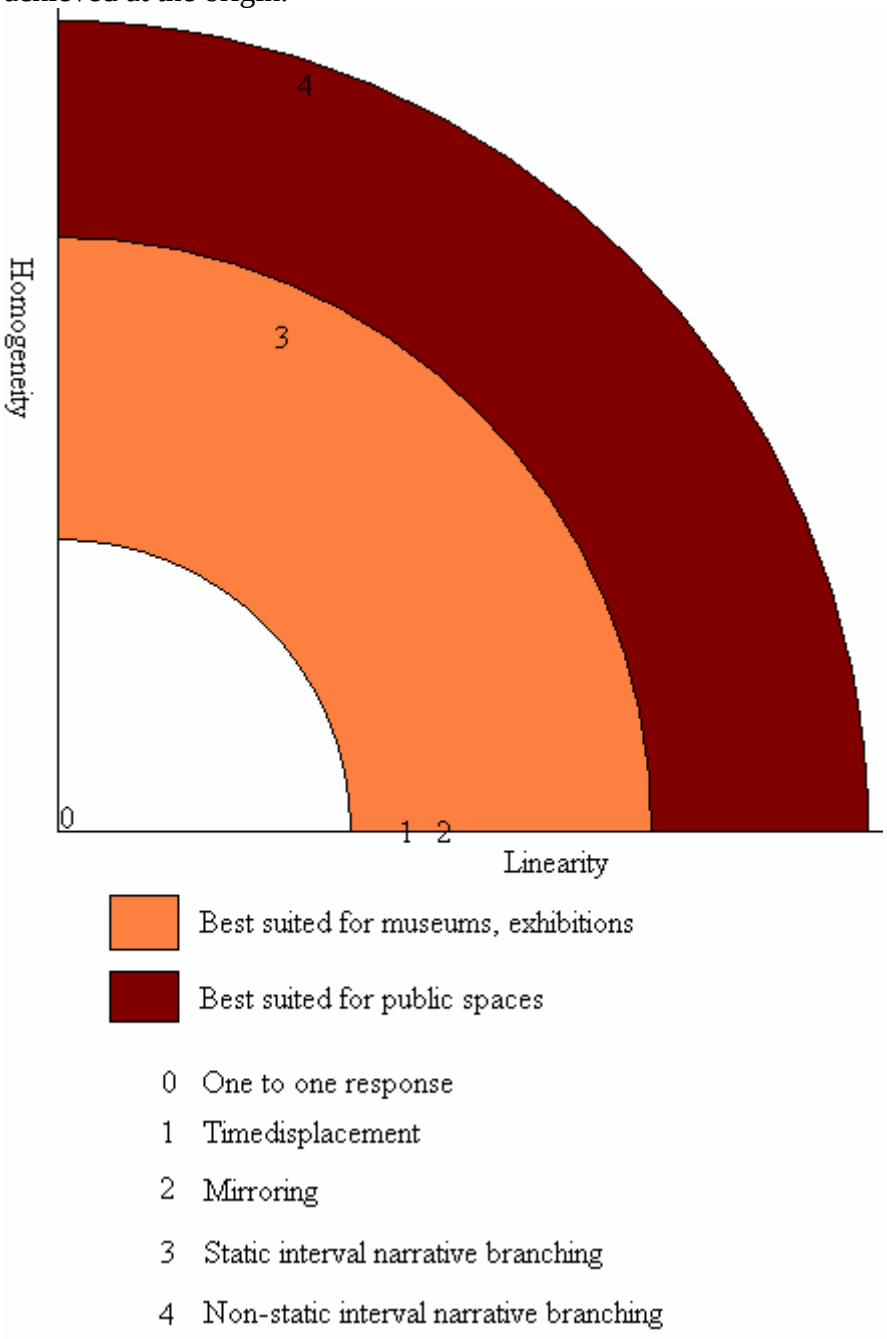

Figure 34 Complexity chart of System Response Patterns

From the interviews and video reviews, it seemed that changes in the narrative structure were considered far more complex in that they took much longer to realize, than direct changes to the way their avatar acted on screen.

This can in part be prescribed to the way the narrative would branch. With video-clips of varying length, the intervals at which changes occurred were not static. Also, the audience has no way of telling that their movements are being stored for later use. Thus it may seem that their input only has an effect at random intervals.

Similar findings were made by Beaudouin-Lafon (2000). He defined three properties of interaction instruments, aimed at gauging their efficiency. Here, interaction instruments are defined as two-dimensional objects containing a physical part in the actual input device, and a logical part in the corresponding ‘tool' displayed on the computer screen. Beaudouin-Lafon proved his interaction model ‘instrumental interaction’ useful in comparing interactivity options in a regular desktop setting. The model however, does not directly apply to embodied interaction due to the lack of logical on-screen counterparts in many embodied interaction systems such as Tapet.

Display locations

With varying degrees of complexity, the different forms of interaction will be best suited for different areas of application. With a higher degree of linearity and homogeneity, the chances of instantly recognizing the aspects of control that reside in the installation are much higher. The downside of this approach is less longevity, which makes the setting of exhibitions or museums well suited for these installations. Audiences will only visit a limited set of times, meaning the instant recognition is important and the audience will not be there to experience the longevity anyway.

With a higher level of complexity, the longevity increases due to much more depth in the interactivity patterns. It could take numerous views just to recognize the first layer of interaction, when ten more could exist. Such installations with a high degree of complexity are better suited to public spaces, where the instant recognition is of less importance. The longevity will mean that the installation does not loose its appeal as quickly.

Figure 34 reflects the ideal locations of use as predicted through the different System Response Patterns. Typically the patterns involving a high degree of non-linearity or very low homogeneity will be better suited for locations that merit frequent visits, while one-time exhibition situations such as museums will favor the opposite.

System Response patterns

By comparing different kinds of communication options our installation provided with the results from the interviews and video analysis, the computerized response to audience input in the form of embodied interaction can be defined as empirically based patterns.

The patterns could be valid with other forms of user input as well, but no studies have been made on this subject.

The patterns may be placed in a chart of complexity as shown in figure 34, using linearity and homogeneity as the unit-axes. Plotted into the chart are the four types of response patterns we used and investigated in Tapet; Timedisplacement, Mirroring, Static interval narrative branching and Non-static interval narrative branching.

Each pattern has a name describing its essence. The name is followed by a few paragraphs of free-form text discussing the pattern, its use and relevance in the case study Tapet.

One to one response

System response comes instantly, and in the most intuitive form. Basically a non-effect pattern, the perfect example of this is how the mouse cursor moves left when the mouse is moved to the left.

One to one response was only used for testing purposes on the audience avatar in Tapet. It is placed at the origin of the complexity-chart.

Time displacement

Input is continuously acted upon in a predictable manner, but the response-time of the system has been increased. Taking the form of a delayed video stream of the audience, the use of time displacement in Tapet was well received among the test subjects. It is quickly recognizable, and provides a stable distortion to the avatar of the user.

Being a low complexity and homogeneous pattern, it is placed along the linearity-axis of the complexity-chart. The Time displacement pattern is well suited to installations aimed at exhibitions and museums, but will probably do well in public spaces as well.

Mirroring

As with Time displacement input is continuously acted upon. The system response is exactly the opposite of expected, hence the term ‘mirroring.’ In Tapet the mirroring was actually a lack of mirroring. When looking in a mirror we have come to expect our image to be a reverse of ourselves, but in Tapet it is not. The mirror-image has been mirrored, canceling the effect.

Mirroring is placed close to Timedisplacement, and has the same expected areas of application.

Static interval narrative branching

Installations using temporal data types such as bits of music, parts of movies or similar may only have branches in the narrative structure at the end of each unit. When each unit is of similar length, the branching occurs at static intervals increasing the predictability. The final version of Tapet used in the third iteration of tests with audience uses this pattern for the base narrative structure. Having removed all but the shortest clips, the difference in duration of the clips lies within five seconds.

The homogeneity of this pattern was perceived as very low, because the audience will not necessarily know that their input is stored and used at a later point in calculating the narrative branching. The sum of input during the completion of the previous unit decides what direction the ‘story’ will take, leaving many different sets of input the same result. Additionally, the same result will not bring about the same branch further down into the narrative structure.

The end result of the summed input however, should be predictable and intuitive leaving the level of linearity fairly high, but homogeneity low. Being of medium complexity, most areas of application should be suitable for this pattern.

Non-static interval narrative branching

As the name suggests, this pattern is a variant of Static interval narrative branching where the units of temporal data types are of varying duration. With the predictability of when narrative branching occurs removed, the homogeneity of this pattern is even lower than that of Static interval narrative branching. This gives it a complexity that is generally too high for installations aimed at one-time visitors such as exhibitions. The total number of different outcomes from input should make it well suited for public spaces.

6.2 Using Java as implementation platform

This section discusses the difficulties and advantages experienced throughout the development process of Tapet using the Java programming language. The three main conceptual components of the system are covered; model, view and controller.

Model (Business logic layer)

Among the three base components of MVC, the model is the one closest to the common database system often attributed to Java. For that reason, the model was expected to be easily implemented and offer a good base for future development of the narrative structure contained within.

Having chosen xml as the format in which to store the logic and having created a simple document structure to contain necessary information, the actual implementation of the model was fairly simple.

Changing the narrative structure of the performance was then contained to altering the representation of the state machine contained in the xml-file, the fine-tuning of which proved an important process that is described below.

Working the narrative structure

Among the more difficult tasks of the development process of Tapet, was specifying the interaction “language” to be used between the audience and the dancer. More specifically, the reaction-pattern of the system was to be decided. To some extent one can design a system and its pattern of response in such a way that it encourages certain user behavior, but in reality no two users will have a truly identical reaction.

Sem, responsible for esthetic design and also the originator of the idea for Tapet, was the one who laid out the original plan for the interaction. With no background in informatics, it was a challenge for her to communicate her ideas in a way that I could easily understand. Her learning to draw state chart diagrams such as the one depicted in figure 9 was a considerable aid in this process.

The descriptor logic for the system response is kept in an xml-file. This works very well with the Java servlet environment, as the Java objects used by the business layer can be instantiated directly from the data contained in the file.

Because all the software layers of the installation are in Java this functionality could easily be added to the presentation layer application, eliminating the need for a separate communication layer. However, keeping the business logic separate and making it available through the HTTP protocol makes it possible to implement a new client at a later point in time, should resources and the need exist.

Initially, the interaction-level in Tapet was intended fairly low. Showing dance in the border-land between a live performance and a mediated performance, thus challenging the stability of these two terms was an essential ingredient. To facilitate this some interactivity was required. The communication between system and user, in this case audience and the dancer, is designed for but not limited to a one-to-one conversation.

If more than one person is present at the set simultaneously, the first to enter will be chosen as target for the conversation and, if possible, kept till the dancer "grows tired" of the given person.

View (Presentation layer)

Of the three software components comprising the Tapet application, the client or view which is termed the 'Presentation layer' in chapter four, was by far the most challenging both in terms of design and implementation.

The early technical design schematics had the chroma keying process performed by hardware and the rescaling and display of the final image done by a small script in Envivio 4Mation. None of these proved possible, and the full task of graphical processing had to be performed by a single Java client.

The final client is pure Java, and uses the JMF extension to access the video content. Two questions stand out after the implementation process was over;

Is it possible to write such a program in pure Java?

The answer obviously, is yes. However:

Is it worth it?

Considerable time was spent optimizing code and finding solutions to problems of different sorts. The major points of interest in these areas are covered in chapter five.

When a program fails to provide the necessary performance, there are two immediate solutions: improve the program, or improve the hardware running the program. With hardware performance increasing near the rate predicted by Moore all those years ago (Wikipedia 2004) the second option is often the cheaper one.

In the case of Tapet, both were in fact chosen. The combination of a faster computer and numerous optimizations made the final version of the client able to provide framerates well above 30, which is more than the filmed material contained. Had the client been implemented using software specifically designed for such a purpose however, both might have been avoided.

The costs associated with finding and learning new technologies able to perform graphical manipulation is not insignificant. Special-purpose high-end software such as 3D-Studio is expensive, and mastering such programs takes time.

If special-purpose software capable of performing the necessary steps as well as the knowledge and skill needed to use it exist, this is clearly preferable. If not, other programming languages, most notably C/C++ can considerably outperform Java. If the option of using these is viable, considerable production time can be saved by avoiding much of the optimization necessary when coding Java. The same applies to programming languages as to special-purpose software however; finding and learning new technologies is costly.

To sum up the previous few paragraphs, using Java for manipulation and playback of audiovisual material is possible, but not advisable if other options are readily available.

Controller (Communication layer)

The need for a separate communication technology first arose with the idea that implementing the view component would not be possible in Java. The view or client as we termed it, was planned implemented using Envivio 4Mation. This however proved impossible and the client was, as explained, implemented with the JMF extension to Java. The decision to use HTTP as protocol for the communication layer was not revoked. The advantages soon became clear, as the computer running the model component was already well loaded with handling the webcam movement detection as well as the servlet. The remote option allowed a separate computer to run solely the client, distributing the workload of the total application over two computers.

6.3 Tapet; a Model View Controller?

The underlying business logic controlling the flow of video-clips and thus narrative structure has much in common with typical Java applications based on small databases. These applications are well suited for the Java language, while the hundreds of thousands of pixels that must be examined, possibly altered and then displayed in the view component are tasks preferably performed by applications written in lower level languages such as C. (Claypool, Coates et al. 2000)

To accommodate the need for an external piece of software responsible for the audio-visual part of the system, the use of a well-known protocol was needed to enable this software to communicate with the application responsible for the logic. The most obvious choice was HTTP. With ports 80 and 8080 open in most networks, firewalls are not likely to stop the traffic. HTTP is also quite common and fairly well supported by Java.

Design patterns or architectural patterns (Larman 2002) can be of considerable help during design, implementation and particularly maintenance/reuse of an application. With the outer technical design decided upon, a suitable design, architectural or user-interface pattern had to be found to facilitate a properly object oriented and reusable set of application components. A search for patterns designed for interactive art provided no results.

Turning to the traditional software design patterns of computer science, the available options were considered.

With the application already broken down into three parts; an audio-visual presentation layer, business logic offering the narrative structure and a communication layer using HTTP, the Model View Controller (MVC) design pattern for its development was considered the best match.

Structured much like a regular GUI application using the MVC design pattern as described in chapter three, Tapet now runs on three main Java classes:

The SceneBuilder class reads, builds and makes available the business logic contained in the xml-file effectively constituting the model part.

TapetServlet works as the controller, acting on input from the motion sensors and the exposed result of the business logic.

Finally, TapetClient works as the view component of the design pattern, displaying the combined results of the input and logic contained in the model part.

As the development progressed however, it became apparent that certain modifications to the MVC structure would have to be made. The MVC pattern is developed for general interface objects where there is a direct coupling between all three components; model

view and controller. The view is updated on request from either controller at input or model on computation, to ensure the latest data are being displayed.

With the view component in Tapet being video-clips modified live, changing view state can only occur at the end of each clip. This again means that neither controller nor model may request a change in view state; view itself will have to request an update when the current video-clip is finished.

Input events are still passed to the controller which stores the events for later use. The controller exposes the model and stored events through the HTTP protocol. When view has finished a video-clip, it will send a request to the controller, which at this point will update the model, process all the events that have been received and send a response. View may then update.

Multi-Narrative Temporal Data based Presentation

This software design pattern used in the development of Tapet, is based on the MVC and named “Multi-Narrative Temporal Data based Presentation.”

It differs mainly from a more common implementation of MVC in view’s lack of responsiveness to the two other components; the view will only be updated by a request from view itself. This is due to the temporal data used in the view component, forcing the rest of the application to await notification from view.

There may still be multiple views per model. However the temporal data type means that only one view may request changes in the narrative structure, or the ‘story’ will run astray. Thus a ‘master’ view is chosen, which will at the end of its current object request the next object, causing the controller to update any other views it may possess as well as the model. Other simultaneous views will be identical except for the fact that they will have to await notification from the controller for updates.

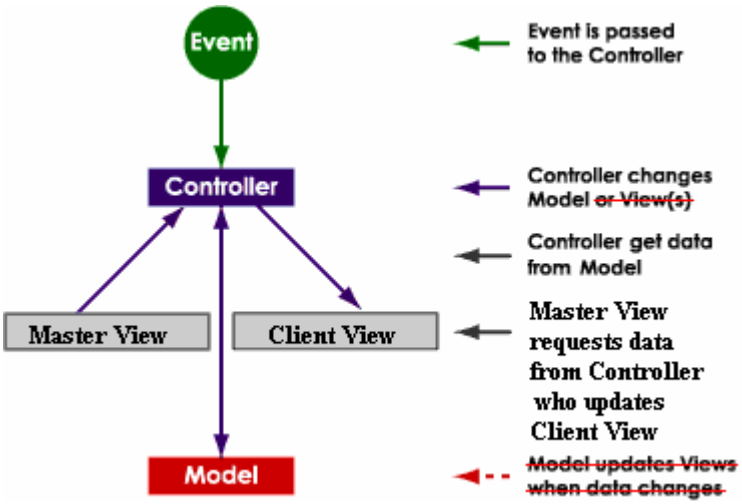

Figure 35 Multi-Narrative Temporal Data based Presentation

The pattern has two sets of events that cause state-changes; the traditional user-input driven event that triggers the controller to update model, and events caused by view finishing its current object. The first type of events causes model to update, and possibly the event to be put in an input-event-queue awaiting notification from the master view.

The second type of events causes the controller to parse through any event queue, request state from model, update any client views and finally return the new object to the master view.

The restriction of view in Tapet is due to the choice of display-technology in the installation; video-clips must be allowed to finish in order to provide the smooth transference between states (video-clips) that is required to provide a feeling of continuity.

This restriction can be generalized to any view component using temporal data. Not limited to video-clips, this could be music, small pieces of software, physical change of state in hardware or the time-out of a web-transaction.

6.4 Summary of discussions

In this chapter the findings related to the two research questions presented in chapter four has been discussed.

The audience experience in relation to the interactivity of the Tapet installation was examined. The understanding of the interactivity, the process of gaining this understanding and the feeling of ‘control’ this gave the audience were identified as the basis for audience perception of interactivity. This basis was divided into homogeneity and linearity, forming complexity as the decisive factor in audience perception of interactivity.

Through the MVC pattern, the software system running the Tapet installation was divided into model, controller and view. Section 6.2 showed that each was possible to implement in pure Java. The model and controller components were found fitting to the Java platform, while the view required more attention to code optimization due to the mediocre performance of Java in relation to speed.

The MVC pattern itself was also found lacking due to the temporal data presented by the view component. To this end, the pattern “Multi-Narrative Temporal Data based Presentation” was developed and used. Section 6.3 showed how it much better suits video and other forms of data that is limited by time.

7 Conclusions

This chapter contains the conclusions to the research questions posed in chapter four, a summary of contributions to knowledge and finally suggestions to future research based on the findings of this thesis.

Based on the identification and description of five System Response Patterns, a conceptual framework describing the system response to embodied interaction in digital interactive art was developed.

The Tapet project was based on the Model View Controller design pattern, separating the three main components; the business logic based on xml-contained data (model), the servlet offering data through the HTTP protocol (controller) and the multi-purpose video-player implemented with Java Media Framework (view).

Implementing the view component in pure Java proved possible, but only after considerable code and algorithm optimization. If other options are readily available, choosing Java would not be advised due to the low performance in relation to speed offered.

With several free and easy-to-use APIs dealing with xml-data, Java proved excellent for the model component. The object-oriented nature of Java proved to offer a sound base for both the organizing, structuring and handling of the business logic of this component.

Due to the possible need of a non-Java view component, the controller of the application was designed to use the HTTP protocol. This ensured a sufficiently high level of separation between model and view to enable different programming languages or applications to communicate with ease. The Java Servlet API made communicating with the model as well as accepting general HTTP-requests easy, thus fulfilling the controller role to a satisfactory degree.

7.1 Summary of contributions

A conceptual framework of interaction in the setting interactive digital art using embodied interaction as input was identified in the form of System Response Patterns. Five patterns were recognized, examined and named during the final phases of the Tapet implementation process. These patterns represent a structured system in the computer-formed output of the interaction-component they represent.

Through the design and implementation of the Tapet installation, it has been proved that an installation of interactive art can be written purely in Java, from the logic to the presentation.

During technical design and the initial phases of implementation the Model View Controller design pattern was used in creating a clean separation between the business logic and the movie-player. Through the process of implementation, a new design pattern based on the Model View Controller aimed at applications with temporal data types in the view component was created. The pattern was named “Multi-Narrative Temporal Data based Presentation” to reflect the narrative structure contained in the previous model component, and the temporal form of the view component.

The implementation of a view component that could perform the fairly complex procedures needed for the Tapet installation video output has been proven possible, but also proved very work-intensive.

The sandboxed environment of a Virtual Machine causes Java programs to run slower than programs made with lower level languages such as C or assembly. To counter the low graphics performance of the Java platform, a highly optimized Java package for handling image manipulation was created. The package, RGBImageTools, contains among others efficient methods for scaling, chroma keying, blur and crossfading. When native Java methods with similar functionality exist, the methods developed are considerably faster.

7.2 Future research

The five System Response Patterns that were identified in this thesis were all used in the Tapet installation, and could thus be classified through the investigation into the effects they had on the audience. Other patterns of similar nature can be found and examined by employing interaction components of a different nature than those used here.

The setting of the Multi-Narrative Temporal Data based Presentation design pattern was in this thesis limited by a model component with temporal data forming a narrative structure; a ‘story.’ Exploring its applicability to other types of temporal data is of interest.

The input paradigm specified in the setting of the System Response Patterns is embodied interaction. It would be interesting to see the influence changing the form of input would have on perceived interaction and experienced complexity of the separate components, as well as on the entire installation.

References

Adamson, C. (2002) Java Media Development with QuickTime for Java. O'Reilly onJava.com (http://www.onjava.com/pub/a/onjava/2002/12/23/jmf.html)

Adamson, C. (2002) Self-Playing Media with Java Media Framework. O'Reilly onJava.com (http://www.onjava.com/pub/a/onjava/2002/10/09/jmf.html)

Adamson, C. (2003) A Gentle Re-Introduction to QuickTime for Java. O'Reilly onJava.com (http://www.onjava.com/pub/a/onjava/2003/05/14/qtj_reintro.html)

Adamson, C. (2003) QTJ Audio. O'Reilly onJava.com (http://www.onjava.com/pub/a/onjava/2003/12/17/qt-bebop.html)

Adamson, C. (2003) Re-Introducing QuickTime for Java. O'Reilly onJava.com (http://www.onjava.com/pub/a/onjava/2003/06/04/qtj_reintro.html)

Adamson, C. (2005). QuickTime for Java: A Developer's Notebook, O'Reilly.

AMODA (2005). Digital Art Definition, Austin Museum of Digital Art.(http://www.amoda.org/about/digitalart.php)

Bannon, L. (1991). From humans factors to humans actors: The role of psychology and human-computer interaction studies in system design. Design at Work: Cooperative Design of Computer Systems. M. K. J. Greenbaum. Hillsdale, New Jersey, Erlbaum.

Barthes, R. (1996). Introduction to the Structural Analysis of the Narrative. Birmingham, Centre for Contemporary Cultural Studies, University of Birmingham

Beaudouin-Lafon, M. (2000). Instrumental interaction: an interaction model for designing post-WIMP user interfaces. SIGCHI conference on Human factors in computing systems.

Bergh, F. v. and V. Lalioti (1999). "Software Chroma-keying in Immersive Virtual Environments." South African Computer Journal **24**(November).

Bezjian-Avery, A., B. Calder, et al. (1998). "New media interactive advertising vs. traditional advertising." Journal of Advertising Research **38**(4): 23-32.

Bordewijk, J. L. and B. v. Kaam (1986). Towards a new classification of tele-information services. The New Media Reader, The MIT Press.

Campbell, J. (2000). "Delusions of Dialogue: Control and Choice in Interactive Art." Leonardo **33**(2): 133-136.

Claypool, M., T. Coates, et al. (2000). Video Performance in Java. Information Resources Management Association Conference, Anchorage, Alaska.

Cycling74 (2005). Jitter.(http://www.cycling74.com/products/jitter.html)

Davison, A. (2005). Killer Game Programming in Java, O'Reilly.

Davison, A. (2005) Playing Movies in a Java 3D World. (http://www.onjava.com/pub/a/onjava/2005/06/01/kgpjava_part2.html)

Day, G. S. (1998). "Organizing for interactivity." Journal of Interactive Marketing **12**(1): 47-53.

Deitel, H. M. and P. J. Deitel (1999). Java how to program. Upper Saddle River, New Jersey, Prentice Hall.

DeVries, J. E. and C. Wheeler (1996). "The Interactivity Component of Distance Learning Implemented in an Art Studio Course."

Dinkla, S. (1994). The History of the Interface in Interactive Art.(http://www.kenfeingold.com/dinkla_history.html)

Dourish, P. (2000). Towards a Foundational Framework for Embodied Interaction, MIT Media Lab.(http://www.dourish.com/embodied/MediaLab.ppt)

Dourish, P. (2001). Where the action is: The foundations of embodied interaction. Cambridge, Massachusetts, MIT Press.

Duncan, S. J. (1989). Interaction, face-to-face; International Encyclopedia of Communications. New York, Oxford University Press.

Eco, U. (1989). The Open Work. London, Hutchison Radius.

enode (2002) Model-View-Controller Pattern. (http://www.enode.com/x/markup/tutorial/mvc.html)

Fagerberg, P., A. Ståh, et al. (2003). Designing Gestures for Affective Input: An Analysis of Shape, Effort and Valence. MUM. Norrköping, Sweden, Norrköping paper

Farlex (2005). Computing dictionary.(http://computing-dictionary.thefreedictionary.com/temporal+database)

Fulk, J., J. Schmitz, et al. (1990). A social influence model of technology use. Organization and communication technology. C. S. J. Fulk. Newbury Park, CA, Sage Publications, Inc.**:** 117-140.

Goertz, L. (1995). "Wie interaktiv sind Medien? Auf dem Weg zu einer Definition von Interaktivtat." Rundfunk und Fernsehen **4**.

Gordon, R. and S. Talley (1999). Essential JMF. Upper Saddle River, New Jersey, Prentice Hall.

Gregory, I. (2002). A Place in History A Guide to Using GIS in Historical Research.(http://hds.essex.ac.uk/g2gp/gis/sect101.asp)

Ha, L. and E. L. James (1998). "Interactivity reexamined: A baseline analysis of early business Web sites." Journal of Broadcasting & Electronic Media **42**(4): 457-474.

Haeckel, S. H. (1998). "About the future of interactive marketing." Journal of Interactive Marketing **12**(1): 63-71.

Heeter, C. (1989). "Implications of new interactive technologies for conceptualizing communication", in Jerry L. Salvaggio & Jennings Bryant (eds.): Media Use in the Information Age: Emerging Patterns of Adoption and Computer Use, p217-235. Hillsdale, Lawrence Erlbaum Associates.

Hieronymi, A. (2004). Myron W. Krueger.(http://classes.design.ucla.edu/Winter04/256/projects/andrew/report.html)

Hoffman, H. (2002) VR Therapy for Spider Phobia. (http://www.hitl.washington.edu/projects/exposure/)

Holden, M. K. and T. Dyar (2002). "Virtual environment training: A new tool for rehabilitation." Journal of NeuroEngineering and Rehabilitation **26**(2): 62-71.

Hutchison, A. (2003). Analysing the Performance of Interactive Narrative. Melbourne, Curtin University

Hutheesing, N. (1993). "Interactivity for the passive." Forbes(12/6/93): 244-246.

IBM (2002). MPEG-4 for JMF.(http://www.alphaworks.ibm.com/tech/mpeg-4/faq)

IBM (2003). IBM MPEG-4 Toolkit.(http://www.alphaworks.ibm.com/tech/tk4mpeg4)

Ishii, H. and B. Ullmer (1997). Tangible bits: Towards seamless interfaces between people, bits and atoms. Conference on Human Factors in Computing Systems (CHI'97), ACM Press.

Jensen, J. F. (1998). Interactivity: Tracing a new concept in media and communication studies, Nordicom Review.

Jensen, J. F. (1998). "Interaktivitet & Interaktive Medier." Multimedier, Hypermedier, Interaktive Medier **FISK series 3**.

Juul, J. (2001). "Games Telling Stories?" Gamestudies **1**(1).

Kearsley, G. and R. Halley (1986). Designing Interactive Software. La Jolla, California, Park Row Press.

Kiousis, S. (1999). Broadening the boundaries of interactivity: A concept explication. Convention of the Association for Education in Journalism and Mass Media Communication (AEJMC). New Orleans

Klein, L. R. (2000). "The joys of interactivity." Library Journal **125**(1): 20-22.

Kurniawan, B. (2001). "Program multimedia with JMF." Javaworld.

Larman, C. (2002). Applying UML and patterns. Upper Saddle River, New Jersey, Prentice Hall.

Lassen, M. (2000). JDOM, The Java DOM.(http://www.webreference.com/xml/column25/)

Laurel, B. (1993). Computers as theatre, Addison Wesley Publishing.

Lee, J.-S. (2000). Interactivity: A New Approach. Convention of the Association for Education in Journalism and Mass Media Communication (AEJMC). Phoenix, Cornell University.(http://list.msu.edu/cgi-bin/wa?A2=ind0101a&L=aejmc&T=0&F=&S=&P=14105)

Leurs, L. (2000). The PICT file format, www.prepressure.com.(http://www.prepressure.com/formats/pict/fileformat.htm)

Luger, G. F. (2002). Artificial Intelligence Structures and Strategies for Complex Problem Solving, Addison Wesley.

Malloy, J. (2000). Narrative Structures in LambdaMOO. In Search of Innovation - the Xerox PARC PAIR Experiment. C. Harris, MIT Press.

McMillan, S. J. (2002). "A Four-Part Model of Cyber-Interactivity: Some Cyber-Places are More Interactive Than Others." New Media & Society **4**(2).

McQuail, D. (1994). Mass Communication Theory. London, Sage.

Neill, J. (2004). Qualitative versus Quantitative Research: Key Points in a Classic Debate.(http://www.wilderdom.com/research/QualitativeVersusQuantitativeResearch.html)

Newhagen, J. E., J. W. Cordes, et al. (1995). "Audience scope and the perception of interactivity in viewer mail on the internet." Journal of Communication **45**(3): 164-175.

Ólafsdóttir, M. E. (2001). Art and Interactivity. The Leading Edge.

Pereira, F. and T. Ebrahimi (2002). The MPEG4 book. Upper Saddle River, New jersey, IMSC Press, Prentice Hall.

Polkinghorne, D. E. (1998). Narrative Knowing and the Human Sciences. Albany, New York, State University of New York Press.

Princeton:WordNet Princeton WordNet.(http://wordnet.princeton.edu/perl/webwn?s=temporal)

QTBLists (2004). QT Broadcaster: maximum *true* resolution?, Apple.(http://lists.apple.com/archives/quicktime-users/2004/Apr/msg00073.html)

QTF-mailing-list. (2002). ""Video frames grabbing"." Retrieved January 5 2006, from http://lists.apple.com/archives/quicktime-java/2002/Feb/msg00062.html.

Rafaeli, S. (1988). Interactivity: From new media to communication. In R.P. Hawkins, J.M. Wieman, & S. Pingree (Eds.), Advancing communication science: merging mass and interpersonal processes. Newbury.

Rafaeli, S. and R. J. LaRose (1993). "Electronic bulletin boards and "public goods" explanations of collaborative mass media." Communication Research **20**(2): 177-197.

Rafaeli, S. and F. Sudweeks (1997). "Networked interactivity." Journal of Computer Mediated Communication **2**(4).

Ryan, M.-L. (2005). Entry: "Narrative". Routledge Encyclopedia of Narrative.

Schultz, T. (1999). "Interactive options in online journalism: A content analysis of 100 U.S. newspapers." Journal of computer mediated communication **5**(1).

Selic, B., G. Gullekson, et al. (1994). Real-time Object-oriented Modeling, John Wiley & Sons.

Shaw, T., K. Aranson, et al. (1993). "The Effects of Computer Mediated Interactivity on Idea Generation An Experimental Investigation." IEEE transactions on Systems, Man, and Cybernetics **23**(3): 737-745.

Sheridan, T. B. (1992). "Musings on telepresence and virtual presence." Presence **1**(1): 120-126.

Smørdal, O. and J. Kaasbøll (1996). Human Work as Context for Development of OO-Modeling Techniques. Method Engineering. Atlanta, Georgia, Chapman & Hall

Sommerville, I. (2001). Software Engineering, Addison Wesley.

Sourceforge (2004). ffmpeg.(http://sourceforge.net/projects/ffmpeg/)

Steuer, J. (1992). "Defining virtual reality: Dimensions determining telepresence." Journal of Communication **42**(4): 73-93.

Stewart, B. (1999) QuickTime for Java. Javaworld (http://www.javaworld.com/javaworld/javaone99/j1-99-bofreports.html#1)

SUN (2002). Java Native Interface using array as a function parameter.(http://java.sun.com/docs/books/tutorial/native1.1/implementing/array.html)

SUN (2003). Image class sumary.(http://java.sun.com/j2se/1.4.2/docs/api/java/awt/Image.html#getScaledInstance(int,%20int,%20int),)

Tolba, O., H. Briceño, et al. (1998). Pure Java-based Streaming MPEG Player. Multimedia Systems and Applications, part of SPIE's symposium, Boston, Laboratory for Computer Science, MIT.

Tyma, P. (1998). "Why are we using Java again?" Communcations of the ACM **1998**(June): 38-42.

UNESCO (2004) What is digital art in Africa. (http://portal.unesco.org/culture/en/ev.php-URL_ID=24376&URL_DO=DO_TOPIC&URL_SECTION=201.html)

Walther, J. B. and J. K. Burgoon (1992). Relational Communication in Computer Mediated Interaction.

Wellner, P., M. Mackay, et al. (1993). "Computer-Augmented Environments." Communications of the ACM **23**(7).

Wiener, N. (1948). Cybernetics. New York, John Wiley.

Wigmore, I. (2005). Whatis.(www.whatis.com)

Wikipedia (2004). "Moore's law entry."

Wikipedia (2005). Digital Art.(http://en.wikipedia.org/wiki/Digital_art)

Wikipedia (2005). Model View Controller.(http://en.wikipedia.org/wiki/Model-view-controller)

Wikipedia (2006). Agile Software Development.(http://en.wikipedia.org/wiki/Agile_software_development)

Williams, F., R. E. Rice, et al. (1988). Research methods and the new media. New York, The Free Press.

www.grack.com (1998). "Java Media Framework."

Young, K. and J. L. Saver (1998). The Neurology of Narrative.(http://www.anth.ucsb.edu/projects/esm/YoungSaver.html)

Zack, M. H. (1993). "Interactivity and Communication Mode Choice in Ongoing Management Groups." Information Systems Research **4**(3): 207-239.

Appendices

Appendix A: JMF supported formats

Supported Media Formats of the Java Media Framework, taken from http://java.sun.com/products/java-media/jmf/2.1.1/formats.html December 26, 2005.

JMF supports audio sample rates from 8KHz to 48KHz. Note that cross-platform version of JMF only supports the following rates: 8, 11.025, 11.127, 16, 22.05, 22.254, 32, 44.1, and 48 KHz.

The JMF 2.1.1 Reference Implementation supports the media types and formats listed in the table below. In this table:
D indicates the format can be decoded and presented.
E indicates the media stream can be encoded in the format.
read indicates the media type can be used as input (read from a file)
write indicates the media type can be generated as output (written to a file)

Media Type	**Cross Platform**	**Solaris/Linux PP***	**Windows PP***
AIFF (.aiff)	**read/write**	**read/write**	**read/write**
8-bit mono/stereo linear	D,E	D,E	D,E
16-bit mono/stereo linear	D,E	D,E	D,E
G.711 (U-law)	D,E	D,E	D,E
A-law	D	D	D
IMA4 ADPCM	D,E	D,E	D,E
AVI (.avi)	**read/write**	**read/write**	**read/write**
Audio: 8-bit m*/s* linear	D,E	D,E	D,E
Audio: 16-bit m*/s* linear	D,E	D,E	D,E
Audio: DVI ADPCM c*	D,E	D,E	D,E
Audio: G.711 (U-law)	D,E	D,E	D,E
Audio: A-law	D	D	D
Audio: GSM mono	D,E	D,E	D,E
Audio: ACM**	-	-	D,E
Video: Cinepak	D	D,E	D
Video: MJPEG (422)	D	D,E	D,E
Video: RGB	D,E	D,E	D,E
Video: YUV	D,E	D,E	D,E
Video: VCM**	-	-	D,E
GSM (.gsm)	**read/write**	**read/write**	**read/write**
GSM mono audio	D,E	D,E	D,E
HotMedia (.mvr)	**read only**	**read only**	**read only**
IBM HotMedia	D	D	D
MIDI (.mid)	**read only**	**read only**	**read only**
Type 1 & 2 MIDI	-	D	D
MPEG-1 Video (.mpg)	**-**	**read only**	**read only**
Multiplexed System stream	-	D	D
Video-only stream	-	D	D

MPEG Layer II Audio	**read only**	**read/write**	**read/write**
MPEG layer 1, 2 audio	D	D,E	D,E
QuickTime (.mov)	**read/write**	**read/write**	**read/write**
Audio: 8 bits m/s linear	D,E	D,E	D,E
Audio: 16 bits m/s linear	D,E	D,E	D,E
Audio: G.711 (U-law)	D,E	D,E	D,E
Audio: A-law	D	D	D
Audio: GSM mono	D,E	D,E	D,E
Audio: IMA4 ADPCM	D,E	D,E	D,E
Video: Cinepak	D	D,E	D
Video: H.261	-	D	D
Video: H.263	D	D,E	D,E
Video: JPEG (420, 422, 444)	D	D,E	D,E
Video: RGB	D,E	D,E	D,E
Sun Audio (.au)	**read/write**	**read/write**	**read/write**
8 bits mono/stereo linear	D,E	D,E	D,E
16 bits mono/stereo linear	D,E	D,E	D,E
G.711 (U-law)	D,E	D,E	D,E
A-law	D	D	D
Wave (.wav)	**read/write**	**read/write**	**read/write**
8-bit mono/stereo linear	D,E	D,E	D,E
16-bit mono/stereo linear	D,E	D,E	D,E
G.711 (U-law)	D,E	D,E	D,E
A-law	D	D	D
GSM mono	D,E	D,E	D,E
DVI ADPCM	D,E	D,E	D,E
MS ADPCM	D	D	D
ACM**	-	-	D,E

Notes:
* - m = mono, s = stereo, c = compressed
ACM** - Window's Audio Compression Manager support. Tested for these formats: A-law, GSM610, MSNAudio, MSADPCM, Truespeech, mp3, PCM, Voxware AC8, Voxware AC10.
VCM** - Window's Video Compression Manager support. Tested for these formats: IV41, IV51, VGPX, WINX, YV12, I263, CRAM, MPG4.

Appendix B: FOBS supported formats

Taken from the FFmpeg documentation page(FOBS builds on FFmpeg):
http://ffmpeg.sourceforge.net/ffmpeg-doc.html
Legacy:

- X = supported
- I = Integer-only version available

FFmpeg supports the following file formats:

Supported File Format	**Encoding**	**Decoding**
MPEG audio	X	X
MPEG-1 systems	X	X
MPEG-2 PS	X	X
MPEG-2 TS		X
ASF	X	X
AVI	X	X
WAV	X	X
Macromedia Flash	X	X
FLV	X	X
Real Audio and Video	X	X
Raw AC3	X	X
Raw MJPEG	X	X
Raw MPEG video	X	X
Raw PCM8/16 bits, mulaw/Alaw	X	X
Raw CRI ADX audio	X	X
Raw Shorten audio		X
SUN AU format	X	X
NUT	X	X
QuickTime	X	X
MPEG-4	X	X
Raw MPEG4 video	X	X
DV	X	X
4xm		X
Playstation STR		X
Id RoQ		X
Interplay MVE		X
WC3 Movie		X
Sega FILM/CPK		X
Westwood Studios VQA/AUD		X
Id Cinematic (.cin)		X
FLIC format		X
Sierra VMD		X
Sierra Online		X
Matroska		X
Electronic Arts Multimedia		X
Nullsoft Video (NSV) format		X

FFmpeg supports the following video codecs:

Supported Codec	Encoding	Decoding
MPEG-1 video	X	X
MPEG-2 video	X	X
MPEG-4	X	X
MSMPEG4 V1	X	X
MSMPEG4 V2	X	X
MSMPEG4 V3	X	X
WMV7	X	X
WMV8	X	X
H.261	X	X
H.263(+)	X	X
H.264		X
MJPEG	X	X
lossless MJPEG	X	X
Apple MJPEG-B		X
Sunplus MJPEG		X
DV	X	X
HuffYUV	X	X
FFmpeg Video 1	X	X
FFmpeg Snow	X	X
Asus v1	X	X
Asus v2	X	X
Creative YUV		X
Sorenson Video 1	X	X
Sorenson Video 3		X
On2 VP3		X
Theora		X
Intel Indeo 3		X
FLV	X	X
ATI VCR1		X
ATI VCR2		X
Cirrus Logic AccuPak		X
4X Video		X
Sony Playstation MDEC		X
Id RoQ		X
Xan/WC3		X
Interplay Video		X
Apple Animation		X
Apple Graphics		X
Apple Video		X
Apple QuickDraw		X
Cinepak		X
Microsoft RLE		X

Microsoft Video-1		X
Westwood VQA		X
Id Cinematic Video		X
Planar RGB		X
FLIC video		X
Duck TrueMotion v1		X
Duck TrueMotion v2		X
VMD Video		X
MSZH		X
ZLIB	X	X
TechSmith Camtasia		X
IBM Ultimotion		X
Miro VideoXL		X
QPEG		X
LOCO		X
Winnov WNV1		X
Autodesk Animator Studio Codec		X
Fraps FPS1		X

See http://www.mplayerhq.hu/~michael/codec-features.html to get a precise comparison of the FFmpeg MPEG-4 codec compared to other implementations.

FFmpeg supports the following audio codecs:

Supported Codec	Encoding	Decoding
MPEG audio layer 2	IX	IX
MPEG audio layer 1/3	IX	IX
AC3	IX	IX
Vorbis	X	X
WMA V1/V2		X
AAC	X	X
Microsoft ADPCM	X	X
MS IMA ADPCM	X	X
QT IMA ADPCM		X
4X IMA ADPCM		X
G.726 ADPCM	X	X
Duck DK3 IMA ADPCM		X
Duck DK4 IMA ADPCM		X
Westwood Studios IMA ADPCM		X
SMJPEG IMA ADPCM		X
CD-ROM XA ADPCM		X
CRI ADX ADPCM	X	X
Electronic Arts ADPCM		X
Creative ADPCM		X
RA144		X
RA288		X
RADnet	X	IX
AMR-NB	X	X
AMR-WB	X	X
DV audio		X
Id RoQ DPCM		X
Interplay MVE DPCM		X
Xan DPCM		X
Sierra Online DPCM		X
Apple MACE 3		X
Apple MACE 6		X
FLAC lossless audio		X
Shorten lossless audio		X
Apple lossless audio		X
FFmpeg Sonic	X	X
Qdesign QDM2		X
Real COOK		X

Appendix C: Interview Guide

Questions posed to the audience in interview sessions during the testing of the finished Tapet installation:

Before entering the installation:
Q1: What kind of interaction do you expect to be possible in this installation?

While test in progress:
Q2: Are you experiencing any form of control over the dancer?
Q3: Is your avatar/image (background video-stream) reacting as you intended?

After test finished:
Q4: Did the level of ‘control’ of the dancer meet your expectations?
Q5: How did the (mis)match between expectation and experience of control affect the quality of your experience?
Q6: What constituted the feeling of interactivity in the total experience?
Q7: How did the delay of the video-stream when you were in the background of the image affect the quality of your experience?
Q8: How did the lack of mirroring when you were in the background of the image affect the quality of your experience?