

# **GAINING CONTROL OF EVENTS IN SITUATED TRAINING SIMULATION**

RE-ENGINEERING PISTE JOURNALIST SIMULATOR SOFTWARE

Jussi Pasanen

BA in Web Media  
Tampere Polytechnic, School of Art and Media  
May 2001

Submitted to the Program in Media Lab  
in partial fulfillment of the requirements for the degree of  
Master of Arts in New Media at the  
University of Art and Design Helsinki UIAH  
September 2003

Copyright © 2003 Jussi Pasanen. All Rights Reserved.



## ABSTRACT

Piste journalist simulator provides practical experience of journalist's daily work to a group of school children in a computer-guided physical-digital environment. Piste is aimed at comprehensive school students from 7th to 9th grade. The simulation takes place in a dedicated space where students spend almost two hours among detailed stage set, interactive scripted characters and compelling personal tasks.

This report documents key aspects of Piste journalist simulator software redesign project. The primary questions are

- How to build a distributed software system that can react consistently to events generated by multiple users in different locations within a physical environment?
- How to connect these events to reactions so that a logical storyline can be conveyed using pre-recorded audiovisual material?

Using a concrete project as an example this report attempts to demystify software design and describe the process in common terms. It shows how user needs are researched and articulated as high-level goals of the solution. These goals are then converted into distinct design tasks. At last it is shown how these design tasks relate to parts of the final technical implementation.

As journalist simulator is a relatively recent phenomenon the report first attempts to define and position it within a larger context. Then, starting from discovering user needs, the design task is defined. The design and implementation of major parts of the software solution are presented. Finally, the product is evaluated against the requirements set in the beginning of the project.

*Keywords:* software design, event system, journalist simulator, hybrid physical-digital environment, interactive narrative.

## **ACKNOWLEDGEMENTS**

I would like to thank Sami Pekkola and the Sanoma project team: Marja-Riika Saaristo, Sari-Maija Marttinen, Pekka Anttila, Heikki Särkelä, and all the wonderful Piste guides.

I also want to thank my instructor Samu Mielonen as well as everyone else who has offered me valuable feedback on this thesis.

Helsinki, 26th September 2003

Jussi Pasanen

# CONTENTS

Abstract	ii
Acknowledgements	iii
Contents	iv
Introduction	1
The project	1
Theoretical framework and context	2
Research questions, audience and scope	3
Research method	3
Report title and structure	4
The Piste journalist simulator	5
Introduction	5
The Piste project	6
Goals of original Piste project	6
A typical visit	7
Instructional method and activities	8
Audiovisual digital environment and story	9
Context and definitions	10
Software engineering and software design	10
Hybrid physical-digital environments and mixed reality	10
Interactive narrative	11
Computer games	12
Interactive fiction	13
Simulations	13
Design task and objectives	15
Evaluating existing system	15
Evaluating simulator software	15
Main goals	16
Users and participants	17
User experience study	18
Requirements	19
Design constraints	20
Solution	21
Planning	21
System and software architecture	22
Development tools and environment	23
Central Control	25
Aggregating user input	33
Story processing	44
Evaluation	56
Product	56
Meeting the requirements	56
Requirements revisited	57
Client feedback	58
Process and people	58
Lessons learned	59
Future development	60
Conclusion	61
Appendix 1 User experience inquiry	62
Appendix 2 Registration sequence	63
Appendix 3 Excerpt of manuscript	65
Appendix 4 Phone server line states	67
References	68
Index	71



Image 1 A view into Piste journalist simulator environment

## INTRODUCTION

### The project

The Piste<sup>1</sup> journalist simulator software redesign project took place in the summer of 2002. Piste is a journalist training environment aimed at introducing the profession to comprehensive school students from 7th to 9th grade. Owned and operated by Sanoma Corporation, it is located in the Sanoma House in the middle of Helsinki. The simulation takes place in a dedicated space where detailed staging, interactive scripted characters and involving personal tasks keep the visitors active for almost two hours. The technical infrastructure of the multimedia program consists of nearly twenty computers, both servers and workstations. There is also a designated phone switch operating the special wireless phones that visitors keep for the duration of the simulation.

The client in the software redesign project was Sanoma Corporation and the new system was supplied by my company Araqua Inc. The project lasted from March to August, and the new system was taken into use in September to November. The primary goals for the project were to *enhance the user experience* of students and guides, and to *increase the operational reliability* of the system. To reach these goals we redesigned and re-implemented the whole software platform, and also made a number of alterations to the scripts of the simulation program.

---

<sup>1</sup> Piste is the name of the Sanoma Corporation journalist simulator. In Finnish, *piste* refers to *period* punctuation mark.

Araqua project team consisted of myself and Sami Pekkola, and we did the design and implementation of the new software system. I was the responsible project manager for Araqua but in addition to this my role varied greatly during the course of the project. My tasks included

- project management
- requirements gathering
- documenting and reporting
- software design on several levels
- software implementation
- database design
- user interface design
- interactive scripting.

Some of these are more relevant from the point of view of this thesis than others, the emphasis here being on software design.

We decided in the beginning of the project that I would take care of server-related development work and Sami Pekkola would be responsible for the workstations. However, we soon realized that such artificial division of responsibilities was unacceptable. As only the two of us were working on design and development, we both had to perform multiple simultaneous tasks at all times.

It is important to note that while I may have assumed a more resolute role in the project the final product is a result of very close and equal teamwork. Sami Pekkola has worked on most of the same tasks as me. Hence, most concepts, ideas or their realizations cannot be clearly credited to either one of us, and therefore they should be treated as shared intellectual property.

## **Theoretical framework and context**

I describe the project primarily from software engineering point of view, with special emphasis on software design. Software engineering is a broad discipline ranging from managing large software projects to the actual specifics of writing production quality code. I attempt to position this thesis between these two extremes and discuss specific software design problems and rationale in the context of the whole project, without getting into the specifics of project management or describing every implementation detail.

Before getting into the software design process I find it necessary to define and position the journalist simulator within a context. As a hybrid physical-digital environment the journalist simulator is hard to classify accurately but I hope to gain some insight by comparing it to other such environments. For instance, mixed reality applications are an interesting point of comparison. Even with seemingly different manifestations, these environments share some common design challenges.

In the journalist simulator the message and lesson are delivered over a story. Because of this I look at the simulator from the perspective of interactive narrative. Interactive narrative scholars attempt to view computers more as a medium instead of mere tools. They make an effort to invent tangible tools to harness the power of computers for telling convincing and touching stories.

The journalist simulator can also be related to game design and research. Although the simulator is deliberately missing many game-like qualities, I find it easy to imagine a highly immersive gaming experience in a similar interactive environment. Game design also tackles many software-related problems that are quite appropriate also for the journalist simulator.

Finally, computer games provide an alternate, more user-centered point of view to software design as opposed to heavily tool- and business-process oriented world of "standard" application software.

Traditional computer-based simulations are one area the journalist simulator naturally relates to, and some ideas can be carried over from other types of simulations.

I discuss all these topics in the Context and definitions chapter (p.10).

## **Research questions, audience and scope**

The questions in the project that I address in this thesis are

- How to build a distributed software system that can react consistently to events generated by multiple users in different locations within a physical environment?
- How to connect these events to reactions so that a logical storyline can be conveyed using pre-recorded audiovisual material?

I describe the *software design process* and solution that arose from these questions in detail. Before that, I attempt to position the journalist simulator environment within context. As a result, the main body of this thesis is formed by the following chapters: *Context and definitions*, *Design task and objectives*, and *Solution*.

I aim this thesis at software and interaction designers and other new media professionals. While reporting the software design process I also attempt to throw light on some of the interconnections between different professions within the field of new media. I believe it is useful for software engineers to learn how user-originated overall goals of an environment are first formulated as design tasks and how these tasks relate to parts of the final technical implementation. In a similar fashion, I think it benefits scriptwriters and interaction designers to know how their visions, stories and models are actually realized in the form of an information system.

As the emphasis here is the software design process I will not cover issues such as digital media project management or client-supplier relationship in detail. Also, I will only superficially touch on topics such computer as an educational medium, or the learning experience of students.

## **Research method**

This thesis is the result of constructive research. Järvinen explains that "it is typical for *constructive research* to build a new innovation and this process is based on existing (research) knowledge and/or new technical, organizational etc. advancements. The utility of the new innovation is sooner or later evaluated." (Järvinen, 2001 p.88) This thesis specifically discusses the design and implementation process and includes evaluation of the finished product. Therefore constructive research appears as a natural research approach.

Järvinen remarks that in constructive research it is essential to describe the process of construction, argue selections and explain decisions, as well as to evaluate the solution and compare it with existing ones (Järvinen, 2001 p.102). In essence, this is what I attempt to do in this report.

This thesis is also informed by case study research, which was originally considered as the principal research strategy. The *case* in this report is the Piste journalist simulator software redesign project. My research questions begin with "how" which favors the use of case studies (Yin, 1994 p.7). However, case studies generally investigate events over which the investigator has little or no control (Yin, 1994 p.9). This is not the case here and therefore I would rather not classify this thesis as a traditional case study.

## Report title and structure

This thesis is titled *Gaining control of events in situated training simulation*. In the first part of the title I attempt to capture four different connotations:

- enhanced system control for the guide (user interface)
- event receipt and delivery (event system)
- event handling and actions (story processing)
- incident presentation (user-perceived story).

The latter part introduces the term "situated training simulation". The word *situated* hints at spatiality and immersion, and the intermingling of physical and digital within the whole environment. As the simulation is closed, i.e. the teaching happens within fixed scenarios, the term *training* was chosen instead of educational. Educational simulators are usually more open-ended. Finally, the *simulation* itself is experiential in contrast with more traditional symbolic simulators (Gredler, 1996 p.523).

I have organized this thesis in the following eight chapters:

I briefly introduce the project and its context in *Introduction*. I also discuss research questions and my research method of choice.

*The Piste journalist simulator* is a more detailed description of the training simulator and its goals.

In *Context and definitions* I discuss the theoretical framework and different phenomena related to the journalist simulator that are relevant in defining the design task and solution.

I evaluate previous software system and describe how requirements for the new system were gathered in *Design task and objectives*.

In *Solution* I describe the processes of conception, elaboration and implementation of the new system, and answer the research questions comprehensively.

I critically compare the new system to its original requirements and goals in *Evaluation*.

I briefly outline some possible directions for future work in *Future development*.

In *Conclusion* I summarize the answers to the research questions.



*Image 2 Students working on the Fresh Kiss case at the news desk*

## THE PISTE JOURNALIST SIMULATOR

### Introduction

Journalist simulator provides practical experience of journalist's daily work to a group of students in a computer-guided physical-digital environment. There is little material available on the history of journalist simulators. Even the name is not well-established, and journalist simulator may also be titled "media center", "mediatek", "mediarium" or "editorial simulator". I prefer the term "journalist simulator" in this paper as it is more descriptive than the alternatives.

Journalist simulator is a Nordic phenomenon with simulator environments installed in Norway, Sweden, Denmark, and Finland. The concept of educating school students in a journalist training simulator was first realized in Norway in 1995 by Expology Burson-Marsteller. They were later on rivaled by another Norwegian company, Media Farm AS. Expology Burson-Marsteller (2002) has created journalist simulators for the following newspaper clients:

- VG "Mediesenter" (Oslo, Norway, 1995)
- Jyllands-Posten "Mediarium" (Århus and Copenhagen, Denmark, 1996-1997)
- Sanoma Corporation / Helsingin Sanomat "Piste" (Helsinki, Finland, 1999)

Expology Burson-Marsteller has also realized the Göteborgs-Posten's media center called "Insidan" in Gothenburg, Sweden in 1996 (Nilsen, 2000 p.84). Apparently Swedish

newspaper Aftonbladet (2003) also sports its own "Mediecenter" in Stockholm (supplier and launch date unknown).

Anders Grov Nilsen's thesis "Utvikling av journalistimulator i Bergens Tidende og Stavanger Aftenblad" is the only reference I could find that directly discusses journalist simulators (Nilsen, 2000). In contrast with my point of view he studies the design process of a *learning environment* using the two journalist simulators created by Media Farm AS as his main subjects: (Ibid. p.6)

- Stavanger Aftenblad "Deadline" / "Mediatek" (Stavanger, Norway, 2001)
- Bergens Tidende "Deadline" (Bergen, Norway, 2001).

## **The Piste project**

Arguably the most sophisticated journalist simulator in the Nordic region is called "Piste". It is owned and operated by Sanoma Corporation, the largest newspaper publisher in the region. Located in the Sanoma House in the middle of Helsinki, Piste is readily accessible from all over Finland.

The concept for Piste was created by Expology Burson-Marsteller (Tietovalta, 1999 p.4). Another Norwegian company 1000 & 1 Digitale Eventyr and an in-house writing crew at Sanoma Corporation wrote the original manuscripts. Tietovalta, a Finnish company, created the production script, graphics, animation and software implementation. Another Finnish company, Studio Bravo!, was responsible for video production and editing.

The cost of the original project was 7.5 million Finnish marks, over 1.25 million euros (Tietovalta, 1999 p.2). The result is a unique, instructive and entertaining training simulator already experienced by tens of thousands of students. Piste was originally launched in the beginning of year 2000.

## **Goals of original Piste project**

The main goal of the Piste experience, and other similar environments, is to promote journalism and critical media literacy in general. In the time of ever-increasing number of different media from which to choose, sustaining existing readership and especially gaining new readers is of vital importance to publishers. While promoting newspapers, the simulator portrays the work of a journalist in a compelling way, making efforts to take also the ethical and moral questions of the work into consideration. Another objective is to make the audience more aware of the processes behind a news story, enabling them to read newspapers in a more analytical way.

Piste is aimed at comprehensive school students from 7th to 9th grade and it attempts to give the audience a concrete example of journalist's day-to-day work. This needs to be done convincingly in a short period of time. The young target audience is lively and challenging. While it is easy to arouse their interest it is much harder to maintain it, especially using conventional means such as lectures or slideshows.

None of the traditional media was chosen for the task. Instead of a leaflet, a slide show, a videotape or a web site, a concrete physical environment was constructed to deliver the message. To animate the environment, a story line and a supporting digital world was created. Together the physical and the digital parts of the experience work towards the same goal, supporting each other. I claim that this is in essence what makes Piste so interesting.



*Image 3 Stage decor, Conducting an interview at the phone booth, Discussion*

## **A typical visit**

Typical visit to Piste consists of the following phases:

1. introduction
2. photographic slide show
3. instruction for simulation program
4. simulation
5. discussion.

In introduction phase the guide leads the student group to Piste environment and discusses general things such as circulation numbers of different Sanoma newspapers. She may also discuss the day's tabloid scoops with the audience, providing a topic of current interest.

The photographic slide show attempts to raise questions and discussion by showing news photographs. The show also contains a story, told by black and white images embedded amongst the color photos depicting actual news events. After the ten-minute show the audience usually has questions or comments about the material. The background and ethical conventions of news images are also discussed.

After the slide show the guide gives brief instructions to the students on how to proceed once the simulation starts. This includes such practicalities as how to answer and place telephone calls and so forth.

The simulation phase lasts approximately one and a half hours, depending on the group size and guide's preferences. The simulation is divided into several parts, first being registration into the system. Most of the simulation takes place at the five separate news desks (see Image 4 on page 9). Each news desk can accommodate six students. In the beginning of the simulation each group of six is assigned a specific news case, i.e. they are given a lead and instructions to write a story. Later on in the simulation students scatter around the environment, searching for information and conducting interviews. The fragments of information they collect are needed later on when the group comes back together to the news desk. The simulation ends with the creation of the newspaper front page.

The visit ends with discussion, chaired by the guide. The details of the simulation program may be discussed, and guide usually has answers to the questions raised during the simulation. Questions such as why a certain choice of headline or photograph is better than another are of special importance, as they serve an educational purpose.

## **Instructional method and activities**

Having witnessed numerous student groups participate in the simulation I would summarize the main contributors to learning in Piste as follows

- participation (actual doing)
- group work
- immersive and stimulating environment.

Participation is the key, making all the difference when compared to the more passive acts of reading, listening or watching. A visit to Piste promotes participation from beginning to the end. For example, students are encouraged to ask questions in every phase. One is much more likely to remember the answer if he asked the question himself.

"Doing" in Piste includes making notes, placing and answering phone calls, conducting interviews and searching for information. Students may get into predicaments, and they are always informed about the correctness of their resolution. All of this is orchestrated so that the students get the feeling of urgency, with constant reminders that one still has to be precise.

Group work is another method of learning in Piste. After collecting information in simulation phase the group needs to discuss and combine the pieces of information each individual has gathered. In the redesign project special attention was paid to add to and intensify these conversations. The amount of time people spend pondering on the questions naturally varies from group to group, and I found it a satisfying experience just to hear how students debate among themselves for quite a while, finally settling on a joint answer.

I would not overlook the impact of immersive environment when evaluating the overall learning conditions. When everything, including the building, physical location, stage decor, lighting and multimedia content support the story being told it is bound to be quite convincing. Having a credible physical environment is a crucial factor in setting up the overall atmosphere of the visit. It is much less likely to have one's mind wandering off the point in Piste than in a typical classroom environment.

Nilsen relates constructivism, situated learning and activity theory to the journalist simulator when discussing the theoretical framework for his thesis (Nilsen, 2000 s.3). Readers interested in journalist simulator as a learning environment and the theories behind that should refer to his paper.

## Audiovisual digital environment and story

In the previous sections I have mostly described the physical aspects of the journalist simulator environment. The digital environment is still more complex and all user activities revolve around it. Controlling the digital environment is the main focus of this thesis.

The digital environment is accessible through a number of *interaction points* embedded throughout the physical environment: doorbells, loudspeakers, touch screens, video monitors, video projector and wireless handsets. These form the physical user interface to the digital world. Digital content is heavily based on audiovisual material; there are hours of video, audio and animation material specifically created for Piste. Videos and animations are presented on touch screens, video monitors and on the video projector. Audio is delivered through loudspeakers and wireless handsets. All of these inputs and outputs together form a user interface that can be described as multimodal<sup>2</sup>.

The whole course of action is orchestrated according to specifically prepared manuscripts. The scriptwriters have had the challenge of keeping the story interesting and even dramatic while still conveying necessary information about journalist's daily work.

The script contains a number of characters that have been specially developed for each of the five cases. For example, in the basic program all five news desks are lead by different editors-in-chief. One very important figure is the special helper character called Beaver. He is a slightly mischievous animated character with wildly varying states of mind. In the end, however, he is always helpful, steering the students to the right direction. Beaver the mascot is portrayed in cartoon-like animation whereas the human characters are displayed in full screen video sequences.

Developing a storyline for situated simulation poses some special challenges. For instance only a single participant can work at a single interaction point at any given time. Writers have had to pay attention to avoid "queues" i.e. situations at interaction points where next student has to wait for the previous one to finish their task before she can proceed. Also, there has been considerable effort in trying to maintain narrative consistency throughout the whole simulation, where the exact path taken by a single student cannot be fully anticipated.

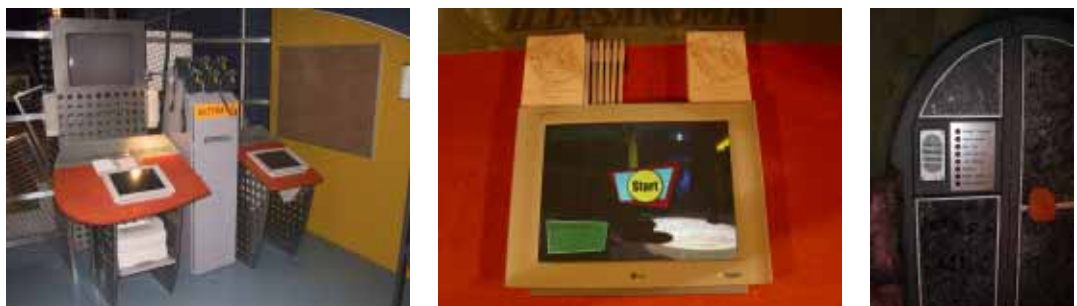


Image 4 News Desk, Touch screen, Doorbells

---

<sup>2</sup> Raisamo (1999 p.2) defines multimodal interfaces as follows: "Multimodal interfaces combine many simultaneous input modalities and may present the information using synergistic representation of many different output modalities."

## CONTEXT AND DEFINITIONS

In this chapter I attempt to position the journalist simulator within a context. Due to the multidisciplinary nature of the realization I find research and disciplines from many different fields relevant to this discussion.

### Software engineering and software design

Software engineering and software design are key disciplines used in the construction of the new Piste software system and they also form the theoretical framework for this thesis.

Software engineering is a determined attempt to render possible the complex process of creating software<sup>3</sup>. Software engineering is a vast field covering such topics as project management, analysis and design, engineering methodology, testing, and quality and risk management. I discuss only the most relevant topics when explaining the design task and solution.

My main point of view in this thesis is that of software design. According to Pressman software design is only about technology: data, architecture, interfaces, and algorithms (Pressman, 1997 p.34). In contrast Blum extends software design to "include all aspects of the software process from the design of a response to a real-world need ... through to design of changes to the product" (Blum, 1996 p.241).

Where Pressman sees software design only as a part of a larger software process with distinctive boundaries, Blum actually makes software design synonymous with *software process*. I am more inclined with the latter view because it emphasizes the real-world i.e. human requirements. In my opinion all software is created to help *people* in one way or another, and this must be kept in mind throughout the whole process of software-making. Blum's take is also interesting in that it makes software design more easily comparable to other design disciplines.

### Hybrid physical-digital environments and mixed reality

The Piste environment combines both physical and digital (i.e. virtual) representations. For example, a series of events may begin with an interview on a touch screen (digital), involve a faxed messages (physical), require checking archives for background information (physical) and need a phone call to a certain number (digital) to finish the task. Action takes place in physical and digital parts of the same *conceptual* environment, advancing the single storyline harmoniously. The boundaries between physical and digital are not necessarily clear to the participant. Also, these boundaries are deliberately blurred in order to focus attention on content and task, not technology.

For the above reasons I would classify Piste as a *hybrid physical-digital environment*. There are interesting examples of other such environments in the European Commission funded

---

<sup>3</sup> A textbook definition of software engineering is offered by Pressman (1997 p.23) who cites IEEE (1993) glossary: "(1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software. (2) The study of approaches as in (1)."

SHAPE research project. SHAPE stands for "Situating Hybrid Assemblies in Public Environments", and among its objectives are (SHAPE, 2003)

- explore hybrid artefacts and the various relationships that are possible between physical and digital manifestations
- examine and construct organized assemblies of hybrid artefacts within room-sized environments as a means for delivering a thematically integrated, yet rich, social experience.

Interestingly, the latter objective characterizes Piste as well. The research vehicles utilized in the SHAPE project range from installations where live video feed is displayed within a real painting (Deus Oculi) (Heath et al., 2001 p.8) to augmented reality interfaces for outdoors (Augurscope) (Schnädelbach et al., 2001 p.38). The main thread of the research discourse is formed by the various design issues related to digital applications that do not operate independently of the physical world but are situated and integrated within it.

Some of the SHAPE deliverables include things relevant to the software design of the new Piste software system. ToneTable, an installation that creates a sonification of virtual objects and participant activity, is one of SHAPE subprojects. Its high level schematic diagram provided inspiration for the conception of our event system (Bowers, Taxen & Hellström, 2001 p.29).

I was originally looking at "mixed reality" projects to find similarities to Piste. According to Milgram (1995 s.1), mixed reality involves "the merging of real and virtual worlds somewhere along the "virtuality continuum" which connects completely real environments to completely virtual ones." Some of the SHAPE projects, such as Deus Oculi and Augurscope, can be easily classified as mixed reality systems.

Mixed reality is mostly demonstrated by augmented reality systems, where user's vision is supplemented with computer graphics. Mixed reality is heavily based on the earlier concept of virtual reality, where literal first person point of view is very important, and implementations usually involve real time 3D graphics. Because of this it was difficult to find results of mixed reality research that would have been directly applicable in our Piste system design.

## **Interactive narrative**

The Piste simulation is structured around a narrative. Although the current manuscripts in Piste dictate that participants cannot really alter the ending of the simulation the sequence of events is changeable. Even if there are pre-scripted paths through the simulation different paths may emerge for instance due to mistakes (student hears the wrong phone number), frustration (student loses track of the plot and starts clicking at random) or willingness of exploration (student wants to see what is "outside" the instructed path). In essence, the participant has limited control over the events that take place.

This kind of unconventional narrative structure has been described as *non-linear*, *multilinear* or *interactive*, among others. The terminology is not well-established and has attracted some fair criticism<sup>4</sup>. The notions of non-linear and multilinear narrative are based on the presumption that traditional narrative is experienced linearly, e.g. from beginning to the end, without skipping or returning to earlier parts of the story. While traditional narrative is typically delivered over a medium (for instance novel) that certainly allows

---

<sup>4</sup> Commenting on media theory discourse Aarseth remarks that "the concept of linearity is at least as ambiguous as that of nonlinearity." (Aarseth, 1997 p.47)

"random access" modern non-linear narratives explicitly require the reader-participant to make choices or take action during the experience.

The Piste simulation is intrinsically interactive. Defining interactivity is no less problematic than defining non-linear narrative. Murray (1997 p.128) points out many of challenges in stating a system "interactive":

*Because of the vague and pervasive use of the term interactivity, the pleasure of agency in electronic environments is often confused with the mere ability to move a joystick or click on a mouse. But activity alone is no agency. ... Agency, then, goes beyond both participation and activity.*

I advocate Laurel's measure of interactivity: "You either feel yourself to be participating in the ongoing action of the representation or you don't." (Laurel, 1993 p.20) Looking at the school children in their predicaments and their joy of discovery in Piste reveals that the simulation is a good example of interactive narrative.

## Computer games

The journalist simulator is sometimes compared to games, although it deliberately lacks some of the qualities of a computer game. Chris Crawford lists four requirements for a game in his seminal book on game design: representation, interaction, conflict, and safety (Crawford, 1982 s.1). While it is hard to say exactly how interactive a game is, or how superficial or profound the interaction, interactivity is still an essential property of a game. Crawford (Ibid. p.12) argues that interactivity is a continuous quality, or a continuum, much like Shedroff (1994 p.10) who discusses interaction design in general.

By safety Crawford (1982 p.14) means that in a game, reality can be experienced safely. Laurel (1993 p.113) talks about the same thing when discussing engagement: "*Pretending that the action is real* affords us the thrill of fear; *knowing that the action is pretend* saves us from the pain of fear." (Original italics.)

The first three requirements of a game are easily satisfied by the journalist simulator: it is representational, interactive, and safe. The one remaining requirement, *conflict*, is described by Crawford (1982 p.13): "Conflict arises naturally from the interaction in a game. The player is actively pursuing some goal. Obstacles prevent him from easily achieving this goal." In the journalist simulator, or at least in Piste, there is little conflict - the simulation steadily progresses towards its predefined conclusion. There is a goal, but it will be reached even if a single member of the group decides not to actively pursue it. There are also seeming obstacles, but they can always be overcome by listening to and following the explicit instructions.

I conclude that the journalist simulator is not a game, even if it has game-like qualities. Calling journalist simulator a game would highlight the recreational aspects too much at the cost of the educational values. In Piste it is emphasized to the audience that it is a simulator, not a game <sup>5</sup>.

---

<sup>5</sup> Curiously enough, Laurel (1993 p.96) comments on a similar tendency twenty years earlier: "[In 1981] A survey of teachers revealed that when a learning game was effective in the classroom, it was reclassified as a "simulation", thus circumventing the categorical problems with games."

## Interactive fiction

According to Buckles (1985 p.8) quoted in Aarseth (1997 p.48) the term *interactive fiction* was coined in 1981. Remembering the issues with the terms "interactive" in general and "interactive narrative" in particular in an earlier section, interactive fiction is considerably better established. While interactive fiction may be anything in which the outcome of a story can be influenced, the term most often refers *computer adventure games* (Granade, 2002). The classic adventure game presents a story complete with a plot, characters, and locations and uses written text as the only input-output medium. Early examples include the original *Adventure* and legendary *Zork* series.

How are these text-only "relics" relevant in the software design of a multimodal heavily audiovisual participatory environment? Surprisingly many aspects of the adventure game engine are of interest when designing journalist simulator software: internal representation of story structure, world rules, and input-output management, to name a few. In truth, many of the more recent adventure games offer semantically much richer interactions with the game world than the journalist simulator, "true" causality, and a more challenging conflict. In the journalist simulator the unfortunate consequence of using pre-recorded audiovisual content is that it makes the story inflexible. Text is more plastic, and it also leaves more space for imagination. Also, it is becoming possible to use the computer to actually generate meaningful real time textual output, making the experience truly different each time. This is not possible with video or spoken narrative just yet.

## Simulations

Simulations are amongst the first conceived applications for computers. Simulation is an "imitative representation of the functioning of one system or process by means of the functioning of another" (Merriam-Webster, 2003). Here "representation" can be replaced with "model" to emphasize that simulations are abstractions of other, usually real-world phenomena.

It is typical of traditional computer simulations to exploit the numerical processing capabilities of most powerful computers to the fullest. This implies one of the difficulties with computer simulations: they are well-suited for simulating physical and other calculable phenomena but much less so for other processes, such as social ones.

Nevertheless, the advent of multimedia and steady decline of hardware cost has made non-numerical simulations a topic of interest. Most interesting simulations for this thesis are *instructional simulations*, which Gredler divides into two principal types: (Gredler, 1996 p.523)

- experiential simulations, establishing a particular psychological reality and placing the participants in defined roles within that reality and
- symbolic simulations, involving the dynamic interactions of two or more variables.

The journalist simulator is clearly an example of experiential simulation, as it satisfies the four essential requirements defined by Gredler (Ibid.):

- a scenario of a complex task or problem that unfolds in part in response to learner actions
- a serious role taken by the learner in which he or she executes the responsibilities of the position
- multiple plausible paths through the experience, and
- learner control of decision making.

Still, it may be questioned whether Piste fulfills the last two requirements. A typical visit to Piste only reveals one of the thirty pre-scripted "paths" to a single participant. Even if this single path cannot be fully anticipated it is quite inflexible. In addition, the participant's decisions do not really affect the later developments in the storyline. Therefore it is doubtful whether the participant truly controls the decision making.

In spite of my previous critical comments simulation is an appropriate categorization for Piste, and I find "journalist simulator" a descriptive name for the environment.

## DESIGN TASK AND OBJECTIVES

In this chapter I discuss the evaluation of the previous system and describe requirements for the new system <sup>6</sup>. This and the next chapter together form my subjective account of the software design process that resulted in the principal parts of the new Piste journalist simulator software.

Before a project or a design task there needs to be a recognized need for improvement. In this case, it was Sami Pekkola's thorough understanding of the original journalist simulator environment and its shortcomings that gave rise to an idea of an improved version. To refine this idea we needed to conduct a more detailed evaluation of the existing system.

### Evaluating existing system

Before starting the actual project we spent considerable time discussing the previous system and its good sides and shortcomings. Sami Pekkola's experience proved invaluable, as he had done considerable amount of implementation work in the original system. More important still was the fact that he has observed large number of student groups work through the simulation. He also has close relations with the guides working in the environment.

As a result of our conversations it became clear that the original system was appropriate for its use and it had high utilization rate. The main problem was that it was not reliable enough. Some of the instability seemed to originate from hardware equipment such as the computers themselves and certain extension cards used to add special functionality. However, it was apparent to us that most problems were software-related, be them glitches in the operating system, drivers, or the special Piste simulator software.

It was in this very early phase that we decided to pursue any enhancements through an upgrade in the software. The hardware and other physical environment were cast as rigid constraints for the design process; we had to work with whatever was already there, hardware-wise.

### Evaluating simulator software

Since a large number of issues in the simulator environment were software-related the existing software system was placed under scrutiny. As we did not have any original design documentation available we had to analyze the source code and external, visible functionality of the system to understand how it works. This is called *reverse engineering*, a "process of analyzing a program in an effort to create a representation of the program at a higher level of abstraction than source code. Reverse engineering is a process of design recovery." (Pressman, 1997 p.794)

The overall system architecture <sup>7</sup> turned out to be a kind of client-server environment with most of the processing responsibility on the clients. The workstation computers exchanged special messages with the main server to control the phone exchange and remote printing

---

<sup>6</sup> "Piste", "journalist simulator", and "system" are used interchangeably in the following text.

<sup>7</sup> See Solution chapter for definitions of system and software architecture (p.22).

for instance. One computer at each news desk was responsible for running the simulation for the related story case. Story state and data, such as names of the participants, were stored on the workstations. The role of the server was primarily that of a message hub, broadcasting messages between workstations and the phone server, and managing print spooling.

In each workstation the functionality was offered by a designated software application. The sequence in which user input was expected and appropriate output generated was programmed in the application. All applications had very rigid structure, i.e. all functionality was inflexibly "hard-coded" in them. For instance, any change to the story, however small, required editing source code in a special development environment and a full recompilation<sup>8</sup>. Also, as each workstation served more or less different purpose, each had a slightly different version of the application installed. With more than fifteen workstations maintenance was difficult, and large number of different programming languages used did not help it.

## **Main goals**

In an early presentation given to the client in March 2002 the following two primary goals were proposed for the project: (Araqua, 2002a p.3)

- enhanced user experience
- increased operational reliability.

What is "user experience" in this case? I see it as the grand total outcome of the simulation; the informational content and emotional effect students get during their visit. Measuring user experience is a very difficult task that requires monitoring users and conducting user tests and interviews.

Of course, only so much of the overall experience can be contributed to with software. What we wanted to accomplish with the new design was a more streamlined simulation with less waiting times and definitely less interruptions due to technical problems. In Pistone guide and system software work in unison to orchestrate the action, and therefore the software should also support guide's work the best possible way.

To reach the above goals we proposed that practically the whole software would be designed and implemented again, i.e. re-engineered. Forward engineering, a concept analogous to re-engineering, is described by Pressman: (1997 p.803)

*The forward engineering process applies software engineering principles, concepts, and methods to re-create an existing application. In most cases forward engineering does not simply create a modern equivalent of an older program. Rather, new user and technology requirements are integrated into the reengineering effort. The redeveloped program extends the capabilities of the older application.*

Even if I had not read it then the above accurately describes what we were planning on doing. Pressman also seems to suggest that re-creating existing functionality, only better, may not justify a re-engineering effort. We had similar thoughts and saw other possibilities of improvement that were not directly related to the software implementation.

---

<sup>8</sup> Microsoft Visual Basic development environment.

Our primary goals were defined in internal discussions and in different meetings with the client. Since we wanted to approach the challenge from the only appropriate direction, the user experience, we decided to conduct a user experience study to find out what kind of problems the new system should address. Before that, however, we needed to identify our users.

## Users and participants

To design any information system it is necessary to know its users first. The system is created to serve its users and not the other way around. I will start by looking at what is *user* in the first place.

Depending on the situation "user" may be too limiting a term to begin with. For example, the "user" of digital television set is also a viewer. Similarly, a "user" of a mobile phone is a caller. For a designer, labeling the audience for the design as "users" has the negative effect of depersonalization. The human qualities of the audience are depreciated. This is especially true in technological determinism, where technology is glorified in and of its own, at the cost of human factors. In a related account Laurel sees another reason to avoid the term "user": "The notion of human agency - the active collaboration of people with programs in the shaping of human-computer experiences - is the reason for my avoidance of the word "user" ..." (Laurel, 1993 p.98)

In an immersive environment user is no longer just a user, but a participant. In Piste, participant is an especially good term, because the experience as a whole involves also many non-digital participatory aspects, such as discussion and group work. Just like virtuality and interactivity immersion can also be thought of as a continuum. The following graph places the distinct user roles, identified below, along this "immersion continuum":

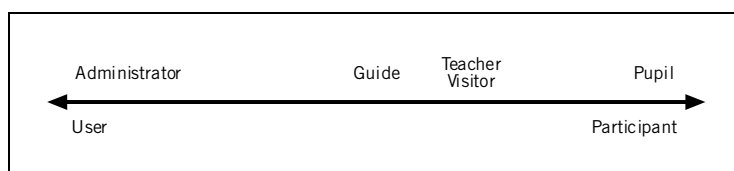


Figure 1 Immersion continuum in Piste

In the Piste redesign project we originally identified the following major user roles: students and guides. *Students* were our main concern, as they are the target audience of the whole environment. To them technology should be as invisible as possible. Students should be protected from having to face potential software or hardware error situations at all cost. Should a problem surface any corrective measures need to take as little time as possible.

Another clear user role was the *guide* who is acting as a host to the student group. From a software feature set point of view, guides have a lot more control over the system than students, and they benefit from having good instruments for monitoring and controlling the simulation state.

We did not explicitly mention the important role of *system administrator* in our original presentation (Araqua, 2002a). We did however propose many aspects of the new system to be designed with administrator in mind.

One potential user group was entirely absent: *teachers* who visit Piste with their classes. They can also be considered users of the system. Although teachers do not normally take part in the simulation they often attend the final discussion, making them participants on a

different level. The members of other *visitor* groups, mostly non-students, can be seen as a distinct user role as well. Teachers and visitors are not particularly paid regard to in the solution, but some functionality is available to them still. For example, any visitor can access the video phone booth and video doorbells completely independent of the state of the main simulation. This is usually used for quick demonstrations.

## User experience study

We identified Piste guides as the most potential source of information regarding the user experience. Guides are in daily contact with the student groups and they have firsthand knowledge on how successful the simulations are. Limiting our direct study to guides also saved us from the practical difficulties of inquiring the students themselves. Our informal questionnaire is in Appendix 1 (p.62). We also used the following sources of information for the user experience study: (Araqua, 2002d p.2)

- Piste visitor questionnaire results, autumn 2001
- Piste journals<sup>9</sup>.

We received written responses to the questionnaire from six guides. We also made an attempt to integrate all relevant verbal input in the user experience study. Our findings, a total of 84 observations, were presented in a 16-page report comprising of the following sections: (Araqua, 2002d pp.3-7)

- technical problems
- observations about script and story
- hardware and setup problems
- required additional guidance
- student group preparedness and attitude
- fortuitous non-reproducible technical problems
- other notes
- case-related notes.

Although the report listed many things that had to be left outside the scope of the project it served its purpose well, providing us a kind of a "checklist" of things to improve on in the new system. As predicted, many of the problems were related to the existing software system, and could be remedied mainly with a new software design. I do not present the full list of findings here due to space and confidentiality constraints but I highlight the most important issues in the next section.

---

<sup>9</sup> The Piste journal is a traditional notebook in which the guides write informal notes every time a group of students visits Piste. A single entry usually contains date and time of visit, home city and size of the group, and any comments about the simulation run or problems encountered during it.

## Requirements

To recapitulate the primary goals for the project were to enhance the user experience of students and guides, and to increase the operational reliability of the system. During the user experience study we identified several requirements for the new system that would help us reach those goals. The most important requirements are listed below:

NO	USER NEED OR ISSUE	REQUIREMENT FOR NEW SYSTEM
1.	Technical problems frequently affect students' work and unnecessarily shift the focus of attention to technology.	The system should be more reliable than the previous one. As a safety measure, in case of an error the system should be easily and quickly recoverable.
2.	Some local (limited to one workstation) problems are reflected to the whole simulation (all workstations).	Workstations need to operate independently of each other.
3.	Students occasionally get frustrated with the "slower" parts of the simulation. Sometimes student groups exceed the time reserved for them.	The overall duration of the simulation should be shorter. It should be possible to start a special 15-minute "alarm case" simulation separately of the basic program.
4.	It is easy for the guide to start wrong simulation by mistake. The state of the simulation cannot be monitored clearly enough. Front pages (the final outcome of the simulation) cannot be shown on the video projector.	The user interface of the guide's workstation should be redesigned. New monitoring functionality should also be added. Front pages should be viewable on guide's workstation.
5.	In case of an error or technical difficulty there is little description or help available. Error history is not available.	System errors should be reported to the administrator in detail and stored for later analysis.

*Table 1 Requirements for the new system*

The list is not exhaustive, but it shows how user needs are translated into requirements. How these requirements further translate into parts of the solution is discussed in the Solution chapter (p.21).

## Design constraints

Design constraints are the limitations within which the proposed design has to operate. We immediately identified hardware as one of the most rigid design constraints in the project. The following diagram shows the hardware environment:

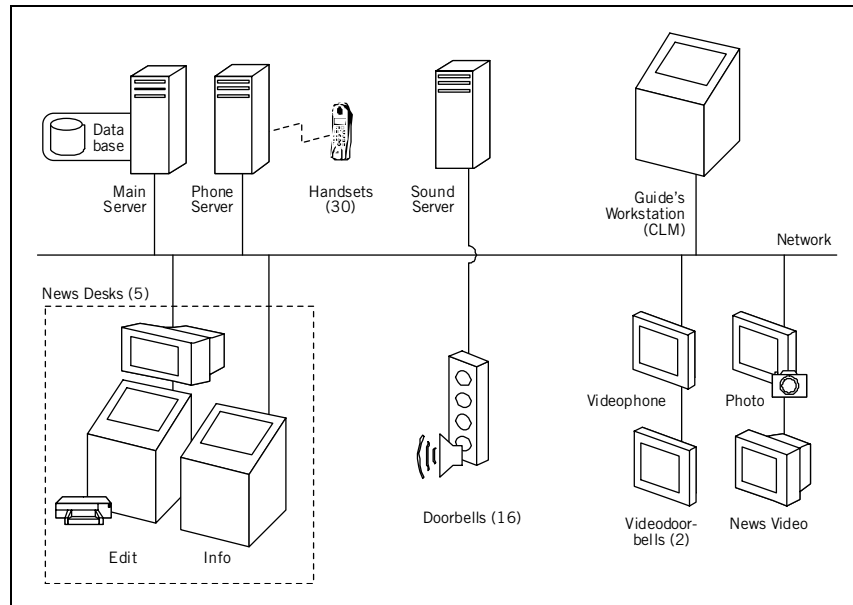


Figure 2 Piste hardware environment

Hardware includes computers, interaction devices (touch screens, handsets etc.), network, and special equipment such as the wireless telephone exchange. Besides hardware the physical environment, for instance staging and lighting, was something we could not alter.

Another key constraint was that the new system should operate using existing multimedia material. We decided early on that creating new video or audio material was outside the scope of the project. While retaining majority of the existing graphics we still needed to make some changes, mostly related to the graphical user interface.

As we were re-engineering an existing system, the starting point for the new one was to replicate previous functionality as closely as possible. Therefore, I also regard original manuscripts as a design constraint for the new system. While it was not our intention to write new scripts, we did eventually modify existing ones and drop some parts of them to make the simulation run smoother.

## SOLUTION

In this chapter I describe key parts of the new Piste system software in detail and answer the research questions comprehensively. I also attempt to present enough evidence so that each part of the solution can be traced back to user requirements and further on to overall project goals.

### Planning

Software projects are traditionally planned by splitting the project into different tasks. The amount of work required by each task is then estimated and the tasks are placed along a timeline. Simplified, that is what we also did with the Piste redesign project. We delivered a very rough and in retrospect overly optimistic version of the timeline in March 2002:

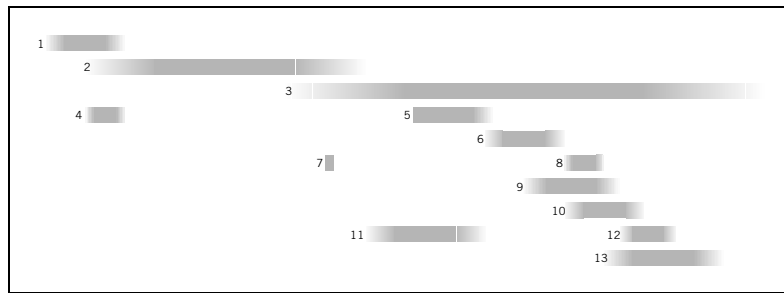


Figure 3 Project timeline, March 2002

This timeline chart (or Gantt chart), like any other diagram, presents an abstraction of reality. Calendar time runs from left to right and the length of each task bar is proportional to its expected duration. Timeline is a planning tool; it is a snapshot of how things are expected to happen. What cannot be shown so easily is the evolution; in our project we had three major versions of the timeline. The March version was further refined and extended in the beginning of the project in late April. To echo the observations from practical work the implementation part of the project (tasks 17.1 to 17.7) was re-evaluated in July:

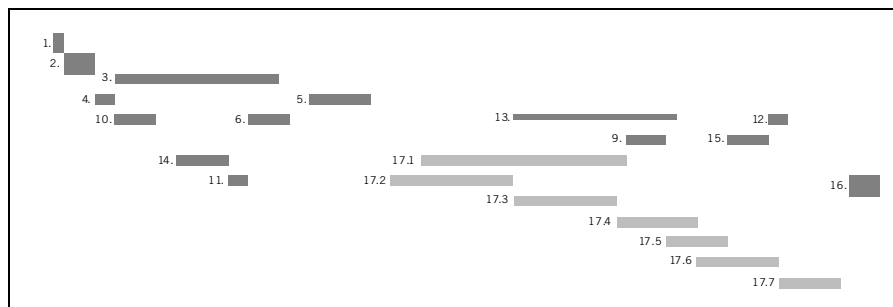


Figure 4 Revised and final project timeline, July 2002

However good an attempt one makes at predicting the future, the uncertainties of design invariably force plans to be refined and reevaluated. Still I think it is of vital importance to do one's best at planning in the very beginning of the project.

As my main objective is to describe the design process instead of doing a strict temporal analysis of project progress I have split up the following discussion according to major *architectural decisions*. For each major part in the system I explain the evolution from user requirement, through architecture and finer design model, to eventual realization.

## **System and software architecture**

In the very beginning of the project we frequently discussed the importance of system and software architecture of the new system. We shared the concerns of Bass, Clements and Kazman (1998 p.31): "Software architecture represents a system's earliest set of design decisions. These early decisions are the most difficult to get correct and the hardest to change, and they have the most far-reaching effects." The same authors define software architecture as follows: (Ibid. p.23)

*The software architecture of a program or computing system is the structure ... of the system, which comprise software components, the externally visible properties of those components, and the relationships among them.  
... a software architecture must abstract away some information from the system ... and yet provide enough information to be a basis for analysis, decision making, and hence risk reduction.*

System architecture is similar to software architecture. The two differ in that system architecture usually regards also some of the hardware components of the environment. I introduce our major architectural decisions for the new system below and elaborate on them in subsequent sections.

Already in our first presentation we proposed that "responsibility should be transferred from workstations to the server" (Araqua, 2002a p.3). After the user experience study it became evident that designing with a client-server model could answer for the requirements of overall reliability (1) and workstations independence (2). The client-server model is the foundation of our system architecture, and as such it heavily affected other architectural decisions. I discuss it in the Central Control section (p.25).

We were intrigued by the "central responsibility" idea so much that we decided to move as much of the processing as possible to the server. Essentially we designated the workstations to be "mindless clients" just obeying the commands issued by the server. This is pretty much the exact opposite of the previous system where majority of processing took place on the workstations. As users would still operate on the workstations, this kind of server-centered architecture required a mechanism for passing and processing user input on the server. An event system was necessary, as I will describe in the Aggregating user input section (p.33). The server-centered design would address the traceability requirement (5) as any errors would be centrally reported and stored.

Yet another early decision was based on our desire to separate story processing from the rest of the system. The story would be centrally processed and maintained on the server, and "manifested" on the workstations through audiovisual and written material. We wanted to avoid the difficulties with the previous system, in which any changes to the story logic required recompilation and reinstallation of workstation software. Later on we decided that story logic should not be realized using same general-purpose programming language as the rest of the system. I will discuss the way story processing was eventually implemented in the Story processing section (p.44).

The requirement for shorter duration (3) dictated that the story itself should be altered. This is not directly related to architecture, but once we could separate story logic from other parts of the software it would be easily modifiable. To address guide's user interface requirements (4) the user interface of the controlling workstation would need to be redesigned, and we should also be prepared to supply the necessary status information and control possibilities with the new server software.

## **Development tools and environment**

Before discussing major parts of the new system in detail I briefly introduce our development environment. One thing that is occasionally overlooked in software projects is the choice of technologies, such as the programming language used, relational database management system, networking middleware, version control system, and so forth. My experience is that it is quite easy to be content with the tools that one knows best, instead of critically evaluating other alternatives. These choices, like early architectural decisions, have repercussions throughout the project. For example, changing the database engine midway can require remarkable modifications to the system implemented so far.

The Piste system software had originally been implemented in a multitude of different programming languages. The client program running on the workstations had been written in Microsoft Visual Basic. This program was responsible for user interface and program as well as story logic. On the server side Microsoft Visual C++, Python and batch scripts were being employed. Furthermore, the designated phone server requires a special language called Parity VOS, which has its own intricacies <sup>10</sup>.

It was clear that the range of languages used should be narrowed down, primarily to reduce overall complexity of the system. After consideration we decided that the new system would be implemented using Visual Basic on the client side and Java on the server side. Visual Basic was an easy choice as we had the source code of the existing system available, and could reuse existing user interface layouts (forms) when required. Also, we had existing Visual Basic components for video and animation playback.

We originally had two candidates for server software programming language: Visual C++ and Java. I had had recent experience developing Java solutions and I advocate Java for its extensive and easy-to-use class libraries, automatic memory management (built-in garbage collector), and portability.

The real question was whether Java could fulfill the performance requirements for the system. A study evaluating Java for game development estimates that Java is 1-2 times slower than C++ (Marner, 2002 p.85). For non-performance-critical applications this is rewarded by 30-65% increase in productivity (Ibid. p.87). As a result of our empirical evaluation, we noted that the server hardware was severely under-utilized having acted mainly as a simple message hub, and came to the conclusion that it should provide enough performance running a Java application.

---

<sup>10</sup> Parity Software VOS is a development environment for creating telephony applications. VOS apparently stands for "Voice Operating System", and it features a structured, Perl-style syntax.

The hardware setup for our development environment consisted of two workstations (one for each of us) and a server. Daily development work, such as writing documents and code, compiling, and testing was done on the workstations. The server was used as a repository, running the following services

- file sharing (Microsoft Windows)
- version control system (CVSnt)
- development database (Oracle).

Version control is of vital importance in any earnest software development process. Although we did not experience any real emergencies during the project, the daily check-out and check-in routine serves as a reassuring indicator of progress, if nothing else.

## Central Control

### The problem

I would like to recall some of the requirements for the new system:

1. The system should be more reliable than the previous one.  
As a safety measure, in case of an error the system should be easily and quickly recoverable.
2. Workstations need to operate independently of each other.
5. System errors should be reported to the administrator in detail and stored for later analysis.

We also recognized that the guides needed to have better monitoring and control facilities available to them, to enable them better do their jobs (e.g. help the students) during the simulation. The responsibilities and roles of different computers were not clearly defined in the original system, which lead to difficulties in system development and maintenance. How could this situation be improved and the above requirements met?

We proposed already in our first presentation that "responsibility should be transferred from workstations to the server" (Araqua, 2002a p.3). What we meant by it was that even if the graphical user interface was to remain on individual workstations, user interface logic and overall control of the system should be on the server.

A centrally controlled system could satisfy all three requirements mentioned above provided that it was designed and implemented correctly. But as in all design, there are other possible solutions to a given task. For example, we were entertaining the idea of a peer to peer system architecture, where each news desk <sup>11</sup> would operate as an independent entity, carrying on its own part of the simulation. Single workstation at each news desk would maintain the story state, and request services from other parts of the system (such as the phone server) when necessary.

The following figure illustrates this imaginary peer to peer system architecture:

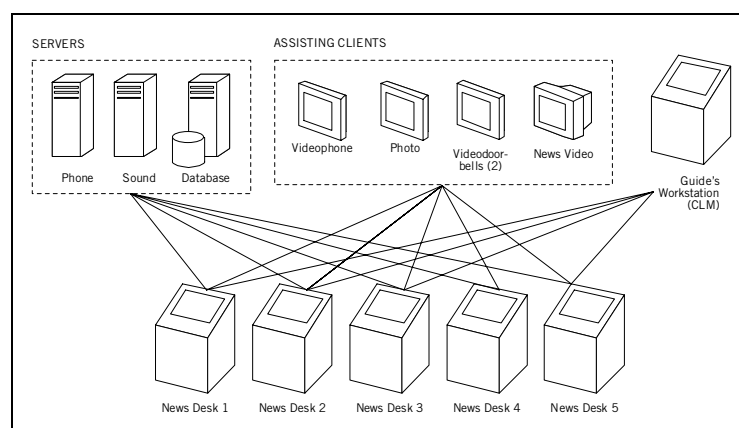


Figure 5 Peer to peer system architecture

As shown in the diagram, the problem with the peer to peer model is the large number of separate connections between different parts of the system. For example, to react

<sup>11</sup> Each news desk consists of two workstations with touch screens and loudspeakers, a TV monitor, and a printer. See Image 5 (p.35).

appropriately to a certain event taking place at the video phone booth, each "edit" workstation would need to maintain direct communication with it <sup>12</sup>.

We also decided very early on that story logic would be separate from the program code and it would be stored in a database. To access this database, each news desk would need its own connection to it. Finally, and most importantly, if simulation state was stored at a workstation as in the original system any problem, such as a system crash, would force the simulation of that news desk to be restarted from the beginning.

Our final system architecture is shown in the following diagram. There are considerably less interconnections between different system components than in the peer to peer model.

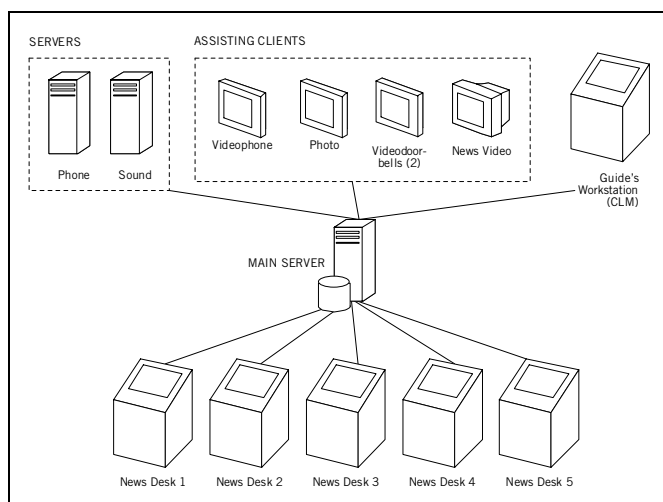


Figure 6 Centrally controlled system architecture

To conclude, our choice of system architecture came down to the question of reliability: are workstations and servers equally reliable and stable, or should we assume workstations are more crash-prone (and therefore less reliable) than servers? We chose the latter, mostly on the basis of earlier experience and intuition, and it was eventually proven that we made the right decision.

### Client-server architecture

A widely used model for designing distributed computer systems is *client-server architecture*. In this architecture, one or more server computers provide certain services for client workstations, usually over a network connection. A typical example is the World Wide Web, where always-on server computers running special web server software deliver internet pages to clients (usually standard workstation computers) on the request of the user. Servers are usually better equipped and more powerful machines than clients and especially their reliability requirements are higher.

The client-server architecture is a development from the times of the mainframe, when all the processing power was centralized in one location, and utilized through simple teletype machines. In contrast to this, in the client-server world both the client and the server participate in the data processing. To recall the example of WWW, even if the server is responsible for storing and serving the pages, it is the client that does the actual rendering of HTML code into a screen image.

<sup>12</sup> For example, once the user finishes an interview an e-mail message arrives for him at the "info" terminal on the news desk.

*Thin client architecture* is a specialization of client-server architecture. The client-side is designed to be especially small so that the bulk of the data processing occurs on the server (Internet.com, 2003). Thin clients often provide only the physical user interface (display, keyboard, mouse) and the actual application software is run on a server that is accessed via network. Depending on the implementation, only the window drawing instructions or screen bitmap data may be transmitted over the wire.

The new Piste software can be seen as thin client architecture. The server is fully responsible for maintaining the story, so the new system can be categorized as a *distributed presentation system* (Pressman, 1997 p.815). This means that while the presentation-interaction component is located on the client, the display is fully controlled by the server.

Also, almost exactly as in Pressman's (1997 p.816) component distribution guidelines, the database is located on the server, and static data (audiovisual material) is retained on the clients. An early schematic of the client-server setup looked somewhat like the following. Some internal components and functionality located within server and client software are also shown.

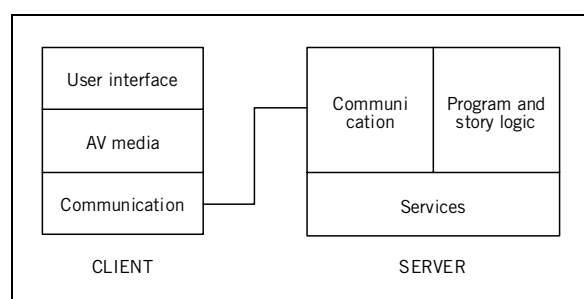


Figure 7 Client-server setup

Similar architectures have been used in other hybrid physical-digital environments. For example, Pirates! is an experiment of "moving computational game elements into the physical world." (Björk et al., 2001 p.1) It is a game that "takes place in a fantasy archipelago setting, where each player takes on the role as the captain of a ship." (Ibid. p.2) The following description of their client architecture highly resembles what we were trying to with our design: (Ibid. p.4)

*The client is essentially a dumb graphics terminal, fully controlled by the server, somewhat similar to how X-Windows functions. The graphics, fonts, and sound files reside on the clients ...*

By having the server fully control the clients one important shortcoming of the original Piste system is remedied: workstations become truly independent of each other. Should a network connection break simulation is resumed as soon as the connection is re-established. Also, if the client crashes for some reason, only a reboot is needed to restore the situation back to where it was because simulation state is managed by the server. This idea of *central control* is fundamental in the new software.

## Communication between client and server

Realizing client-server architecture requires means of communication between client and server. This is called a communication protocol. Protocol may be defined as follows: (Everything2, 2003)

*A definition of rules of communication between two or more nodes through a medium. The nodes can be computers, people, countries, or other things. The medium can be speech, [postal] snail mail, a network or other kind of wire, or anything else that can carry a signal or message.*

We decided very early on that the common TCP/IP protocol suite would be used as a foundation of all network communication. But TCP/IP only offers a way of reliably establishing and maintaining a communication channel between two computers over a network. The actual content of the messages that are being sent back and forth have to be defined in an *application-level protocol*.

Most importantly, the application-level protocol defines the *vocabulary, syntax* and *rules* for the messages transmitted. It also defines the responsibilities of both client and server. Although it was clear to us that there was not any ready-made protocol that would satisfy all the needs of our new system, a few technologies for managing specifically the syntax of the protocol were evaluated: XML-RPC and SOAP.

### Evaluating technologies for protocol syntax

Put shortly, XML-RPC<sup>13</sup> allows one machine to execute some computational function on another machine via a network. The syntax of the requests and responses is based on XML<sup>14</sup>. For the purposes of our project we identified the following benefits of XML-RPC: it is open, simple and XML-based. It also has implementations available for both Java and Visual Basic.

However, from our point of view XML-RPC had severe disadvantages:

- XML-RPC operates as a stateless protocol, i.e. a TCP-connection is opened by client to send one request and receive one response. This behavior is not well suited for continuous operation with frequent passing of messages.
- XML-RPC assumes that client always initiates connection. In our system, it is often necessary that the server commands the client into another state, for instance to display another user interface form.
- XML-RPC adds unnecessary HTTP headers in the conversation that slow down operation.

Another protocol alternative was SOAP, or Simple Object Access Protocol<sup>15</sup>. SOAP has excellent implementations available on both Microsoft and Java platforms, and it has similar benefits as XML-RPC: it's open and XML-based. Unfortunately, all the same disadvantages apply to it, too. Additionally, and contrary to its name, SOAP is actually relatively complicated, making use of advanced techniques such as namespaces, versioning, and complex data types.

---

<sup>13</sup> More detailed definition of XML-RPC is available at its home page at <<http://www.xml-rpc.com/>>. RPC is an abbreviation for Remote Procedure Call.

<sup>14</sup> XML is short for Extensible Markup Language, a metalanguage for describing other languages. XML allows the design of customized markup languages for different types of documents and applications. XML is a World Wide Web Consortium Recommendation.

<sup>15</sup> World Wide Web Consortium's definition of SOAP is available at <<http://www.w3.org/TR/SOAP/>>.

## Principles and vocabulary of communication protocol

Based on the evaluation results stated above, we decided to design a proprietary protocol for the purposes of the project. The original principles and rules for the protocol were as follows: (Araqua, 2002c)

1. the client initiates a TCP-connection with the server
2. the connection is used bi-directionally
3. messages are encoded in simple text or XML syntax
4. messages sent by the client to the server are called *events*
5. messages sent by the server to the client are called *commands*
6. the client always acknowledges the receipt of a message
7. the server never acknowledges the receipt of a message.

These were refined later on. We decided to use XML syntax and rewrote rule 3 as "messages are encoded in simple XML syntax". We also dropped rule 6, effectively eliminating all (unnecessary) acknowledgement messages in the protocol.

Besides the basic rules of a protocol, the actual vocabulary is very important. To come up with the necessary commands and events, we performed a sequence analysis of the beginning of the simulation program. It was good working with an existing system because it was relatively easy to extract the necessary information by observing how it functioned.

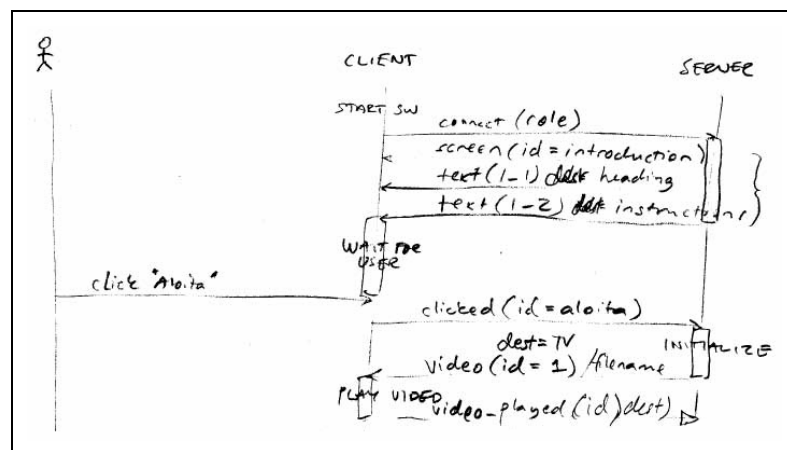


Figure 8 Beginning of registration sequence

We used simplified *sequence diagrams* to document the user registration phase of the original system. The beginning of the registration sequence is shown in the example above. Sequence diagrams are specific kind of interaction diagrams in the UML<sup>16</sup> notation. Fowler and Scott define sequence diagrams as "models describing how groups of objects collaborate in some behavior" (Fowler & Scott, 1999 p.67). As the name implies, sequence diagrams give emphasis to the sequence of interactions. Our original diagrams concentrated on documenting user input and system responses. They can be found in Appendix 2 (p.63).

<sup>16</sup> The Unified Modeling Language is the industry-standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems. It simplifies the complex process of software design, making a "blueprint" for construction. (Everything2, 2003)

## Early communication protocol

For the first version of the protocol we labeled messages as *events* (client to server) and *commands* (server to client). The following tables define the early protocol.

NAME	PARAMETERS	DESCRIPTION
clicked	<i>name</i> , button name	User clicked on a button.
connect	<i>role</i> , client role	Client has connected to the server. First message after initiating connection.
register	<i>name</i> , user name	User registers with the system.
userAnswered	<i>id</i> , unique user id <i>filename</i> , name of sound file that was played	User has answered and listened to a call (system to user)
videoPlayed	<i>id</i> , screen id <i>filename</i> , video file name	Playback of video or animation file has ended.

Table 2 Event messages in early communication protocol

NAME	PARAMETERS	DESCRIPTION
callUser	<i>id</i> , unique user id <i>filename</i> , name of sound file	Call user and play a sound file.
screen	<i>name</i> , name of screen (form)	Show another screen.
text	<i>id</i> , text string id <i>position</i> , text position name	Show text string at specified position.
video	<i>filename</i> , video file name <i>destination</i> , screen id	Play video or animation file.

Table 3 Command messages in early communication protocol

The following example shows how these events and commands may translate to actual over-the-wire messages:

client	<code>&lt;connect role="edit1" /&gt;</code>
server	<code>&lt;screen id="introduction" /&gt;</code>
server	<code>&lt;text id="1_1" position="heading" /&gt;</code>
server	<code>&lt;text id="1_2" position="instruction" /&gt;</code> (user clicks "start"-button)
client	<code>&lt;clicked id="start" /&gt;</code> (system is initialized and the case starts)
server	<code>&lt;video id="tv" filename="1_1.avi" /&gt;</code> (video finishes)
client	<code>&lt;videoPlayed id="tv" filename="1_1.avi" /&gt;</code> ...

Table 4 Example of protocol messages

The communication protocol underwent many changes and extensions during the project. Names of events and commands changed, and many more were added. A third message type was introduced, called *control* messages (client to server), required for specific functionality on the guide's workstation. The final system has close to 30 different messages, each with a number of specific parameters. Still, the original design goal of simplicity is present in the final implementation.

## Unifying client software

One downside of the original implementation was that almost all client machines required a different client executable. Maintaining such a large number of different versions had proven difficult and cumbersome, and had frequently resulted in version control conflicts where corrections or modifications of the software were not included in all client executables.

Along with the choice of the thin client architecture we decided to create a unified client executable. In the new system the client software is identical across all workstations, and how a workstation behaves is defined in a configuration file. This effectively enables replacing any workstation with another one in live situation, on the presumption that similar hardware is available and required audiovisual material is copied over to the new machine.

## Support servers

In addition to the main server and client workstations there are two additional servers in the system:

- phone server
- sound server.

Phone server is responsible for controlling the designated phone switch that operates the wireless handsets. Sound server manages the doorbell-speaker combinations, playing back appropriate sound files when users push specific doorbell buttons. We originally strove for keeping these two servers outside the scope of the project.

### Sound server

The only function of the sound server is to relay static information (speech) to the users. Doorbell responses need not be dynamically altered, i.e. same button always results in playback of the same audio file. Also, the progress of the story does not depend on receiving information when users actually push doorbell buttons. Therefore, doorbells and respective speakers operate independently of the rest of the system, and no communication occurs between main server and sound server. However, we did prepare in our new system design for adding this connection later, should it turn out that a future revision of the simulation program requires more detailed information about user interaction at doorbells. To summarize, we did not need to alter any of the sound server software.

### Phone server

In contrast to the sound server phone server is such a crucial component that we eventually had to integrate it with the rest of the new system. This need became apparent quite late in the project. The original system used a combination of proprietary C++ software, batch scripts, and VOS code to control phone calling and answering. Message passing was done by changing the contents of different text files which clearly was not acceptable for the new system. To complicate matters further a necessary piece of hardware was missing<sup>17</sup>.

By studying the VOS documentation we found out that TCP/IP-networking support could be added with a special add-in. By adding networking capabilities to the phone server we

---

<sup>17</sup> The phone software development environment requires using a *dongle*, a special hardware protection key. This key needs to be connected to the computer whenever new phone server software is compiled or run.

could treat it as yet another client, even though a specialized one. We received a replacement dongle from Norway and could carry out first hands-on tests with the VOS language - a language we did not have any previous experience with.

The original phone server software architecture consisted of a single *main task*<sup>18</sup> that creates a separate *line task* for each phone line (e.g. handset). We retained this architecture with the new system. We cleaned up the main task code and added TCP/IP networking functionality to it. The new main task acts as a message hub, creating a network connection from phone server to main server. It delivers incoming and outgoing messages between main server and individual line tasks.

The line task code was rewritten from scratch. Each of the separate line tasks spends most of its time in a traditional message loop<sup>19</sup>, where incoming messages (from main server to phone server) are only handled in certain states. Probably the most challenging part of writing the new phone server software was designing and implementing a suitable *state machine*<sup>20</sup> for the line task. The distinct states and transitions are listed in Appendix 4 (p.67).

For handsets and the phone server, another application level protocol was developed. Since the issue of phone server was tackled relatively late in the project we had a more refined version of the client protocol available. This made phone server protocol design relatively straightforward. We used the same categorization of messages to *events* (phone server to main server) and *commands* (main server to phone server). For example, to call user's handset main server issues a *call command* to phone server. Similarly, when a user attempts to dial a number, phone server sends *dialed event* to main server, so that appropriate response audio file can be looked up in the database.

---

<sup>18</sup> VOS is a multi-tasking environment, which means that one or more tasks can be executed concurrently. VOS task is roughly equivalent to the concept of a thread in Java and other programming languages.

<sup>19</sup> Message loop is a basic concept familiar to most user interface programmers. In message loop, the program basically watches an input queue for any messages. If the queue is empty nothing happens. As soon as a message arrives, it is picked from the queue and handled appropriately.

<sup>20</sup> Finite state machine (FSM) is a kind of digital circuit, that is used to process information in steps, or states (Everything2, 2003). State machines are discussed in more depth in Modeling behavior: state machines section (p.52).

## Aggregating user input

### The problem

We had decided that we would be creating a centrally controlled system, and we knew that it should closely replicate the functionality of the original system. Not yet concerned with the actual responses, we still recognized the obvious need of responding to user actions in a meaningful way. This is called processing, and may be modeled as follows:

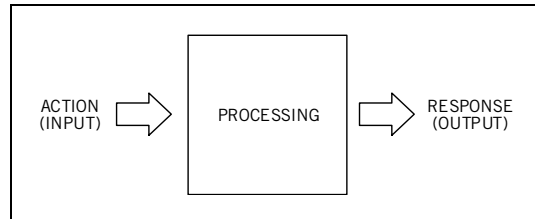


Figure 9 Computational processing

User action (input) invokes processing that produces a response (output). Information systems have traditionally been designed in terms of input, processing and output. In the early era of computing most software operated sequentially. Input was specially prepared for the software, usually in a computer file, which the software then read, did all necessary processing, and stored results in an output file. Many software programs still operate in this non-interactive way.

However, this sequential model is not suitable for creating interactive, graphical user interfaces. We wanted to create a system where certain *input on one device* could generate desired *output on any other device*. Instead of processing within a single computer, we needed a way to send user input from different workstations and aggregate it on the server. Server would then do the actual processing, and command appropriate clients to generate perceptible output. To solve this problem we first needed to identify all possible input and output devices.

## Physical user interface

To document the environment we created an inventory of all digital equipment and their physical locations (Araqua, 2002b). The following map displays the positions of all interaction points in the system:

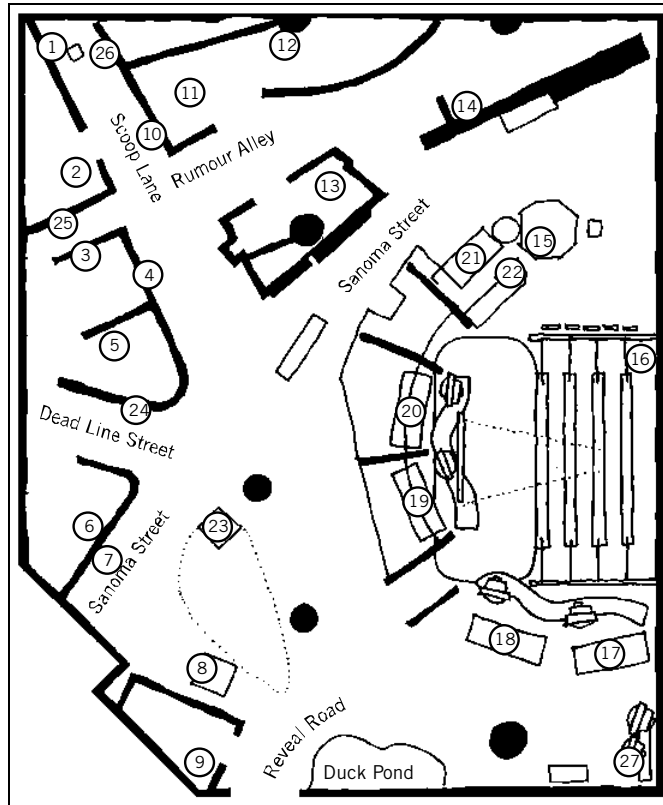


Figure 10 Piste floor plan with all digital interaction points

Numbers 1-16 represent doorbells, 17-21 news desks (edit and info workstations), 22 is guide's workstation and 23-27 are assisting clients. For software design purposes the physical user interface may be summarized in terms of input and output capabilities of each device:

DEVICE	INPUT	OUTPUT
touch screen	press	graphical user interface graphics and animation
TV monitor	-	video (full screen)
projector	-	video (full screen)
speaker	-	speech and audio (with touch screens) speech (with doorbells)
printer	-	printed document ("fax")
handset	dial (user to system)	speech (system to user)
doorbell	press	-

Table 5 Device input and output capabilities

All of these devices are either directly or indirectly connected to the information system, as can be seen in Figure 2 (p.20). The most important types of user input are presses on touch screen, dials and answers on the handsets, and doorbell presses.

There are some additional devices present in the environment that are not connected to the information system. For example, an occasional task for the students is to "take pictures" with a physical camera secured to a stand located in front of a computer monitor. The camera shutter button is not connected to the computer, so in reality the activities at the camera do not affect the simulation in any way. However, we did design the new system so that in the future adding new devices like the camera is relatively easy.

The following image shows one of the five identical news desks. A news desk consists of one *edit* workstation, one *info* workstation, and six handsets. Edit workstation, shown on the left, has touch screen, TV monitor, speakers, and a printer attached to it. Info-workstation on the right is equipped with a touch screen and a pair of speakers.



Image 5 News desk

### Graphical user interface

In Piste, the graphical user interface is maintained by the workstations and accessible on the touch screens mounted at several locations throughout the environment. By analyzing the original system we concluded that the most important graphical user interface elements are

- buttons
- text labels
- animations
- background graphics

Most of the views<sup>21</sup> of the graphical user interface can be constructed of the elements<sup>22</sup> listed above. The following figure shows an example of actual graphical user interface view, and the corresponding form with different elements shown as wireframes:

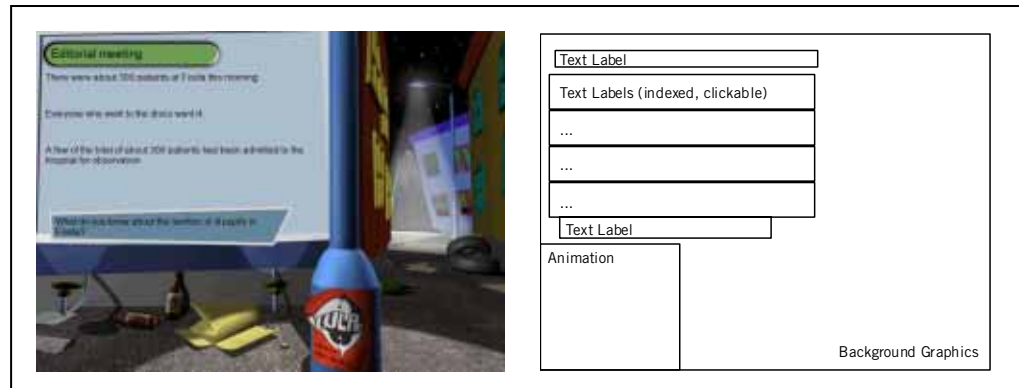


Figure 11 Graphical user interface

More complex widgets of modern GUI's, such as lists, scrollbars or hierachical trees are not used. We decided to use the same programming language for client application as was used in the original system, Microsoft Visual Basic.

#### Perils of Visual Basic

Visual Basic is a powerful environment for creating user interfaces. All the standard Microsoft Windows user interface controls are easily at author's disposal. The basic frame in which the controls are placed is called "form" and it closely resembles a normal window. Adding a simple button is as easy as dragging its icon from the toolbox onto the form. Behavior can be added by double-clicking on the button and writing the desired code in a full-fledged programming language, Basic.

The danger of Visual Basic is in this seeming easiness of creating software. The coupling between external representation (user interface), internal logic (program code) and data is very tight. It is characteristic of Visual Basic that user interface and program code are actually stored in the same source files. Visual Basic authors have to pay special attention to structure their programs appropriately.

In the original Piste system, the processing logic was embedded in the user interface forms inside the Visual Basic application. In the new system, all such logic needed be located in the server. This called for a way of communicating user actions from client to server, and the results of those actions right back to the client.

<sup>21</sup> View means one distinct screen in the user interface. In Visual Basic views are called "forms."

<sup>22</sup> Also called controls, or widgets.

## Delivering user input to server

To deliver necessary input information to the server we rely on an application level protocol. Simplified, the clients only gather relevant user input and relay it to the server over the network.

In the graphical user interface we are mostly interested in button and text label controls. As they may be clicked on by the user, they form the core of user input from the touch screens. Since we need to know which button was clicked we rely on button's name (referred to as *id*) in the protocol. The early draft of this can be seen already in our first sequence diagrams in Appendix 2 (p.63). Sending button names instead of for instance coordinates across the network has clear benefits: it is on a higher level of abstraction, and it makes the traffic more easily human-readable. The latter is especially useful in the development phase.

## Event systems

The traditional sequential software model is not sufficient for graphical user interfaces, or interactive software systems in general<sup>23</sup>. Fortunately, there are other models, or architectural styles, to support more direct interaction between users and software (see Bass, Clements & Kazman, 1998 p.94).

A widely used model with graphical user interfaces is the *event driven model*. In event driven model, each user input (mouse click, keystroke) can be detected and acted upon by the program. Most modern operating systems with visual interfaces, such as Microsoft Windows and Macintosh OS X, implement event driven model in some form. Most visual software development environments, such as Visual Basic<sup>24</sup>, are also based on this model.

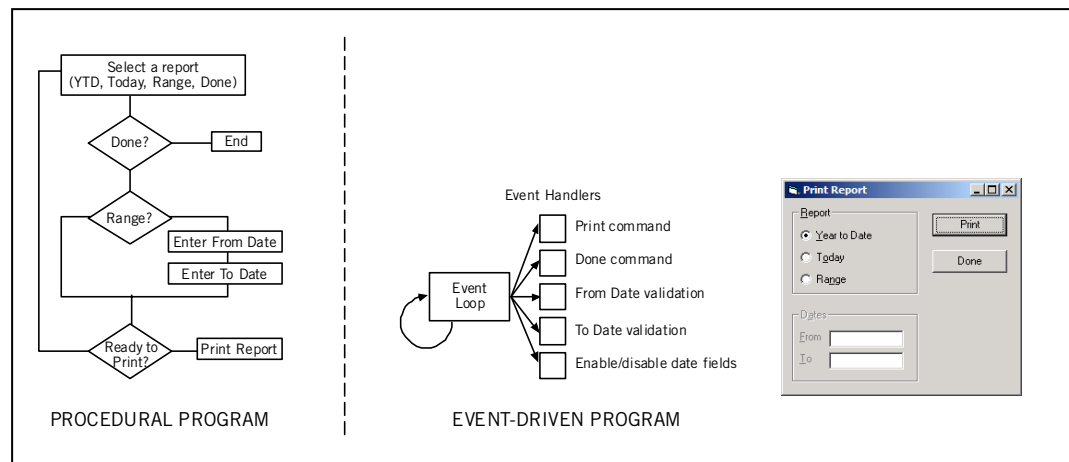


Figure 12 Procedural vs. event-driven programming (Adapted from Microsoft, 1999 p.11)

In procedural software the main program controls how information is entered by the user (Microsoft, 1999 p.11). In contrast with this, an event-driven program allows the user to enter information in any order. Also, user actions invoke smaller code segments (Ibid.). These code segments are often called event handlers, or event procedures.

<sup>23</sup> Sequential software model is also called structured, or procedural model.

<sup>24</sup> In Visual Basic, a piece of program code that is executed when a given event occurs is descriptively called an *event procedure*.

Our earliest sketch of how user input could be turned into events is presented in the following figure:

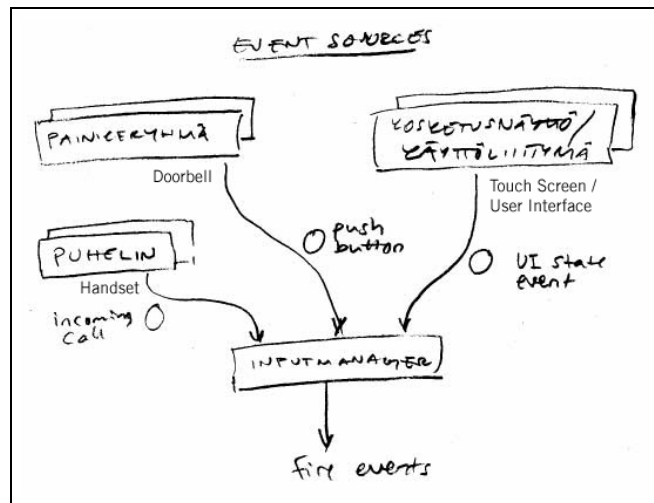


Figure 13 Event sources and input aggregation

As shown all input is centrally aggregated and turned into events. Each event contains information about respective event source and other necessary details so that it is ready for further processing.

#### Patterns and idioms

In software engineering, models and patterns are important tools for representing and transferring knowledge. Model is an expression of a design. According to Fowler and Scott (2000 p.34)

*Patterns describe common ways of doing things. They are collected by people who spot repeated themes in designs. These people take each theme and describe it so that other people can read the pattern and see how to apply it.*

Patterns are by no means restricted to the domain of software engineering, quite the contrary. For example, patterns are documented and used extensively in architecture. Patterns are a way to avoid reinventing the *idea* of the wheel. That is, even though two design (or engineering) problems differ in details, both problems and their solutions have similar enough structure, so that an earlier solution may be applied to a later problem.

Whereas patterns are rather generic by their very nature, there are also "common ways of doing things" within a more limited, or specific, domain. Regarding programming languages Venners describes the difference between design pattern and idiom as follows: (Venners, 1998)

*The difference is one of scope. In general, both design patterns and idioms describe solutions to recurring design problems. But with a design pattern, both the problem and solution are generic enough to be independent of implementation language. An idiom, by contrast, is a low-level pattern that is specific to a programming language.*

The term "idiom" does not have nearly as wide an acceptance as "pattern". Nevertheless, as Venners' article considerably inspired the design of our event system, I describe his "event generator idiom" in the next section.

## The event generator idiom

Event generator idiom is a specialization of the observer design pattern<sup>25</sup>. The intent of the event generator idiom is to "Enable interested objects (listeners) to be notified of a state change or other events experienced by an "event generator." " (Venners, 1998)

Venners (Ibid.) describes a simple scenario where an event occurring at the telephone (ringing) is of interest to another device, the answering machine. Referring to real-world objects, his example and model are close to actual software implementation of such a system. For explanation consider the following diagram:

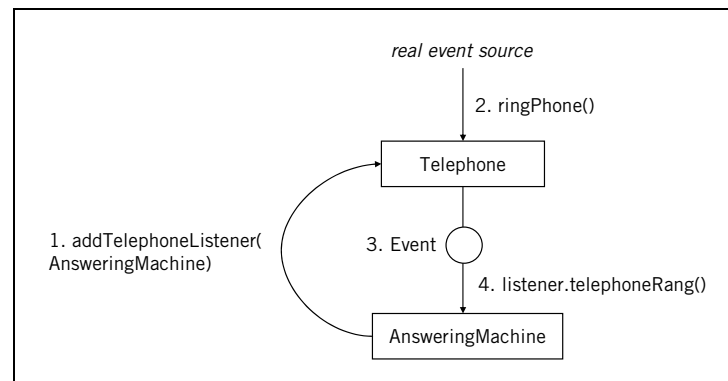


Figure 14 Telephone and answering machine event system

First, the answering machine expresses its interest in receiving ringing event notifications from the telephone. Answering machine does this by registering with the telephone (1). An outside stimulus is applied to the telephone (dialing) (2). The telephone creates an event object with relevant information (for example date, time, and caller id) (3). Telephone then looks up its list of registrations, and sends this event to interested parties (4). AnsweringMachine is invoked through a designated interface (`telephoneRang` method).

According to Bass, Clements and Kazman (1998 p.102) event system is an important paradigm

*because it decouples implementations from knowing each others' names and locations. As mentioned, it may involve decoupling control as well, which means that components can run in parallel, only interacting through an exchange of data when they so choose.*

For us parallel processing was important, because even though we had a single server controlling all activities and doing all processing, it was necessary that different clients and their respective functionality on the server operated independently of each other.

## Device classes

A class is an OO [object-oriented] concept that encapsulates the data and procedural abstractions that are required to describe the content and behavior of some real world entity (Pressman, 1997 p.572). Classes and interfaces are the basic building blocks in object-oriented software design.

Early on, we decided to model most important physical devices as distinct classes. This allows them to act as event generators (event sources) whenever user input takes place. Where applicable, the device class also supplies an interface for sending messages to the corresponding real-world device, enabling the system to produce perceptible output.

<sup>25</sup> Observer pattern is also know as the publisher-subscriber pattern.

The final Device class and its subclasses are shown in the following diagram. Some methods are left out for clarity.

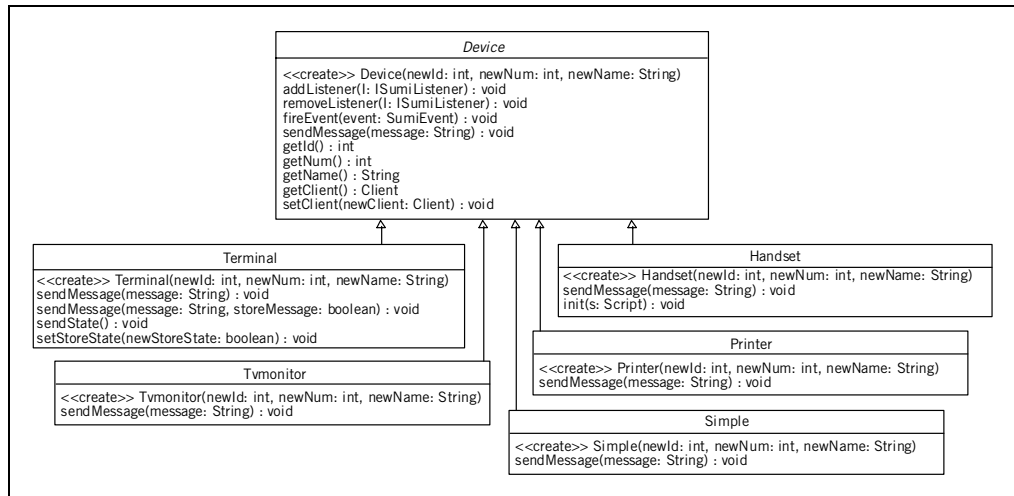


Figure 15 Device class and subclasses

The abstract Device base class enables each device type to act as an event source. For example, a listener may subscribe by invoking the `addListener` method, and unsubscribe by invoking the `removeListener` method of a device. Each device keeps track of those class instances that wish to receive event notifications from it.

Device class and its subclasses are described in the following table:

CLASS	DESCRIPTION
Device	Abstract base class for all input and output devices. Provides functionality common to all devices.
Terminal	Represents a single workstation. Supports storing outgoing messages in a <i>state buffer</i> .
Tvmonitor	Represents a TV monitor for playing full screen video.
Handset	Represents a single handset.
Printer	Represents a printer device for printing fax pages.
Simple	Represents a generic device.

Table 6 Device class and subclass descriptions

#### Event class and related interfaces

The event itself is also modeled as a class (see following figure). Each event has type, name, and attributes. An event is created by an event generator, i.e. a device<sup>26</sup>. To act as an event generator a class has to implement a specific interface, `IEventSource`. As the Device base class implements this interface all subclasses automatically inherit the behavior. Similarly, to receive events a class has to implement an interface called `ISumiListener`.

<sup>26</sup> In the actual implementation Event instances are usually created by the Client class after it has received and interpreted a message from an actual client over the network.

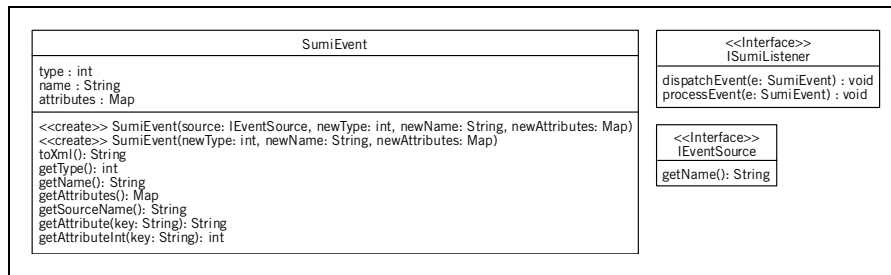


Figure 16 Event class and related interfaces

A short example is in place. In the beginning of the simulation, a member of a student group is required to press a "start" button displayed on a touch screen. To see how this action is delivered to interested listeners is displayed in the following diagram:

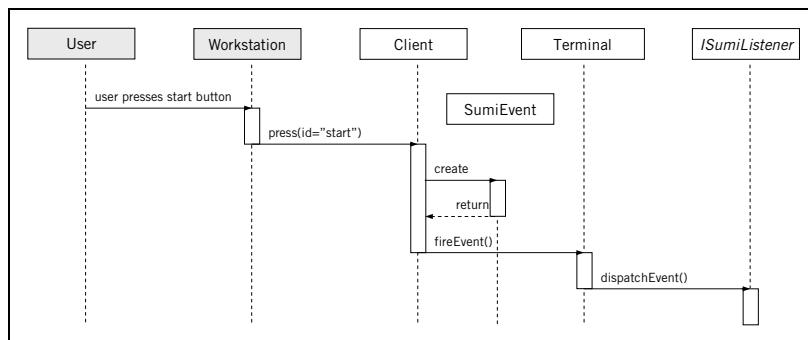


Figure 17 Button click sequence diagram

When the button is pressed (by user), the client application (running on the physical workstation) detects this and delivers a *press* message (with button id "start" attached) to the server. On the server, client message is received by Client class, which in turn creates a SumiEvent instance of the occurrence. Each Client class has a corresponding Terminal class (see previous section) that gets invoked using its *fireEvent* method. Terminal class, in turn, refers to its list of listeners, and delivers the event to all interested classes by calling their *dispatchEvent* methods.

Exact contents of the event depend on the device in question. In the preceding example, the protocol message is "`<press id="start" />`". This translates into an event named "press" with type *user*. Event's attribute map contains a single entry named *id* holding value "start".

### Concurrent processing and event delivery

One critical aspect of the new system was that it must support concurrent processing, i.e. different parts of the system need to operate independently of each other. Multitasking and parallel processing in general are some of the most difficult aspects of software design. Having multiple threads running simultaneously presents many dangers and challenges that software designers need not worry about in conventional single-threaded applications.

According to the Java Tutorial, "a thread - sometimes called an execution context or a lightweight process - is a single sequential flow of control within a program. You use threads to isolate tasks." (Sun Microsystems, 2003) In our system we use several concurrent threads to allow the server to interact with multiple clients simultaneously. We also use threads to isolate different parts of the story called "cases" from each other.

How are threads related to the event system? Invoking a class method happens in the execution context of the calling class. Therefore, whenever a Device invokes a listener by calling its `dispatchEvent` method, the listener is obliged to return immediately. It cannot block, i.e. do any long-standing processing.

As the processing time required in different parts of the simulation varies considerably, we needed a way to decouple event source and actual processing of events from each other. To solve the problem the concept of *event queue* was introduced. Event queue is actually just another design pattern that was quite appropriate for our need. Java environment itself relies extensively on event queues in user interface event processing. The following diagram shows on the left how Java Abstract Windowing Toolkit (AWT) events are queued by the Java runtime system:

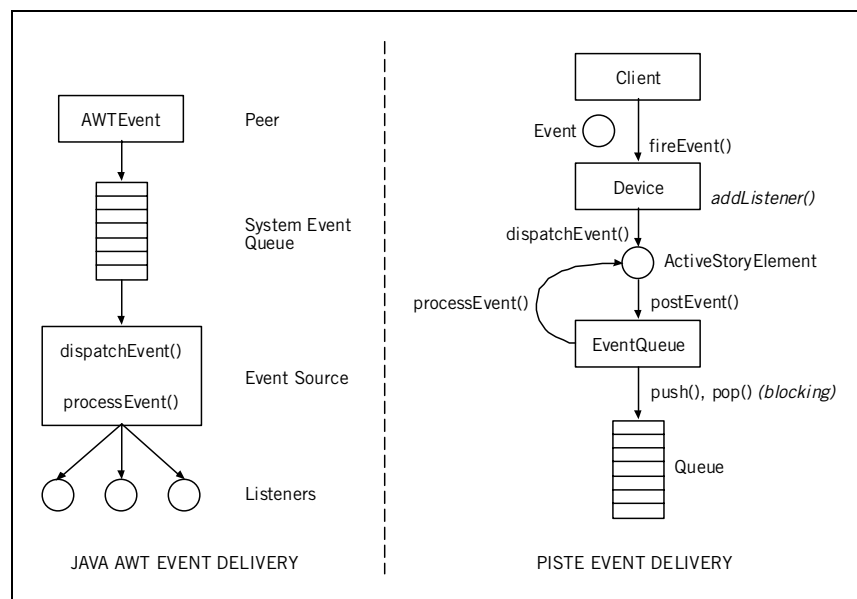


Figure 18 Java AWT event delivery compared to Piste event delivery

Java AWT system event queue holds *all* user generated events, such as mouse moves, clicks, and key presses. It operates as a single, global queue.

After recognizing the need for an event queue we attempted to modify the system event queue for our purposes. To make use of it we would have needed to aggregate all events from all devices into the one queue. While possible this has the drawback that each listener would have to scoop through all incoming events, most of which are irrelevant to its function. What we needed was one event queue for each device. Our final event queue design is shown in the previous figure on the right.

Compared to the Java AWT event delivery mechanism we added more details underneath the listener. In Figure 18 on the right, `ActiveStoryElement` is the listener, interested in receiving event notifications from `Device`. When events occur, they are delivered to the listener through its `dispatchEvent` method, as before. However, instead of immediately processing the event (i.e. acting upon it), the event is posted to an `EventQueue` (`postEvent` method). After posting, the control is returned back to `Device` right away. The execution context of `Device` is therefore free to continue dispatching events to other listeners, or do other processing.

Meanwhile `EventQueue`, running in its own designated thread, notices that an event has been posted to its queue. `EventQueue` also holds information about related listener. It picks

up and removes the event from the queue, and calls listener's `processEvent` method. Listener may spend any amount of time to process the event, as processing now happens within `EventQueue`'s execution context. When finished, control is returned back to `EventQueue`, which subsequently processes any pending events in a similar fashion. The Device itself is at all times free to dispatch more events to the queue, without paying any regard to possible long-standing processing caused by earlier events.

Our event queue model does not guarantee that events "immediately" reach interested listeners. That would require a different architecture, more like those used in real time systems, with rigorous response time requirements. Our event queue model does, however, guarantee that every event is delivered to its destination. This is very important considering the way story processing is designed. I discuss this in detail in the next chapter.

## Story processing

### The problem

The Piste story is a narrative of events that is manifested through video clips, animations, speech and textual material. Unlike a conventional written story, in Piste users have control over the order of different events. The most interesting challenge in the new Piste software was creating an "engine" that receives user input aggregated by the event system, consults a programmatic version of the storyline, and responds appropriately.

Recalling another requirement (3) we had to make the following improvements in respect to the earlier system:

- The overall duration of the simulation should be shorter.
- It should be possible to start a special 15-minute "alarm case" simulation separately of the basic program.

We also wanted to make the guide's work easier by unifying the story structure of the five separate "cases". This means that overall series of events is similar in all of the five cases.

We started working on the problem of the story engine from two different directions. First, to determine how exactly should the system respond (i.e. output) to certain actions we studied the original system. As with the communication protocol, our early draft showing the sequence of events in the registration phase of the simulation was of great help (see Appendix 2, p.63). Second, we needed to model the script to be able to make a programmatic version of it. We started by analyzing the structure of the original manuscripts.

### Analyzing story and script structure

To get an idea of the story some background information about the cases is necessary. Each self-contained case represents creating a single news story. The five cases run concurrently and independently of each other. Each one of the original manuscripts is about 100 pages long, and they were produced by a team of writers from 1000&1 Digitale Eventyr and Sanoma Corporation.

One of the cases, titled "Fresh Kiss", is used as an example in the following discussion. The synopsis of the story begins as follows: (Expology, Sanoma Corporation & 1000&1 Digitale Eventyr, 1999 p.2)

*301 schoolchildren in the suburb of Foxila developed a rash after a school disco. Initially, an outbreak of mononucleosis or 'kissing disease' was suspected, but it soon became apparent that the cause of the rash was the Fresh Kiss toothpaste that had been distributed at the disco.*

What is described is the key event, the beginning of a story that kicks off the whole case. All subsequent events relate to that in one way or another, as students try to figure out what lead to the incident. The final outcome of the simulation is a newspaper front page, complete with heading, ingress and a photograph.



Image 6 Students working on the Fresh Kiss case at the news desk

### Case structure

The order of events on typical Piste visit is as follows:

1. introduction
2. photographic slide show
3. instruction for simulation program
4. simulation
5. discussion.

In the simulation phase the overall structure of each case follows identical pattern. By analyzing the manuscripts we came up with an early draft showing the pattern:

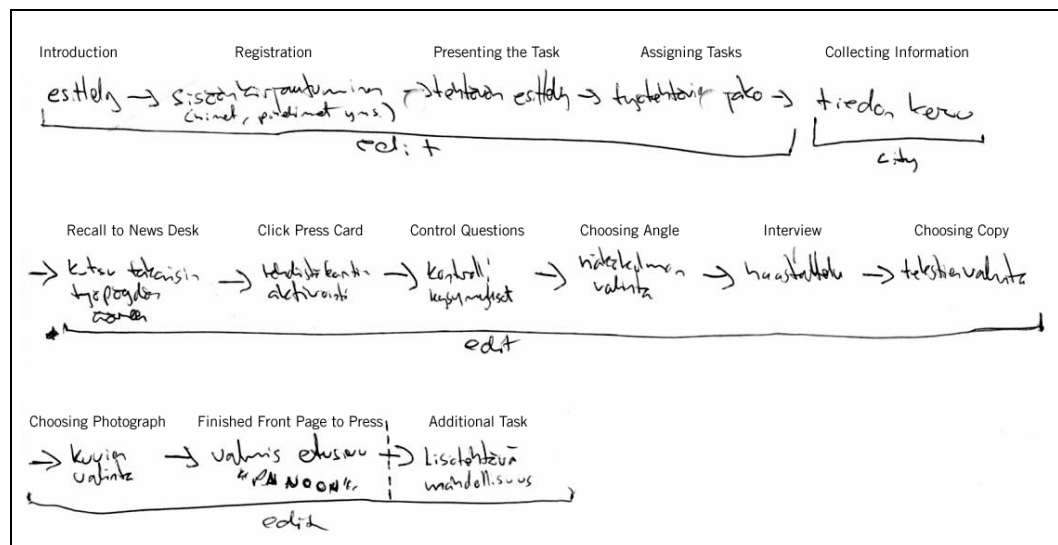


Figure 19 Breakdown of case structure

In the final system this translates into the following sequences:

NO	SEQUENCE	DESCRIPTION
1	Registration	Each student enters her name into the system and picks up her handset to receive a call from Beaver (the helper character).
2	Individual Tracks	Students scatter around the environment to search for information and conduct interviews.
3	Editorial Meeting	Students "discuss" the case with editor-in-chief by pondering on the questions presented and choosing appropriate answers.
4	Choosing Angle	Students are presented three alternative headlines and they have to choose the correct one.
5	Interview	Together the students conduct a video interview with an important character in the story to gather final pieces of information for the scoop.
6	Choosing Front Page	Students are presented three alternative ingresses and they have to choose the correct one.
7	Finished Front Page	Students are shown the final front page image complete with headline, ingress and photograph.
8	End	End of simulation.

Table 7 Case structure and sequences

#### Individual tracks structure

With the exception of individual tracks, all action takes place at the news desk *edit* terminal. Individual tracks sequence is of special importance - another early sketch attempts to display how user "paths" separate when they enter that sequence:

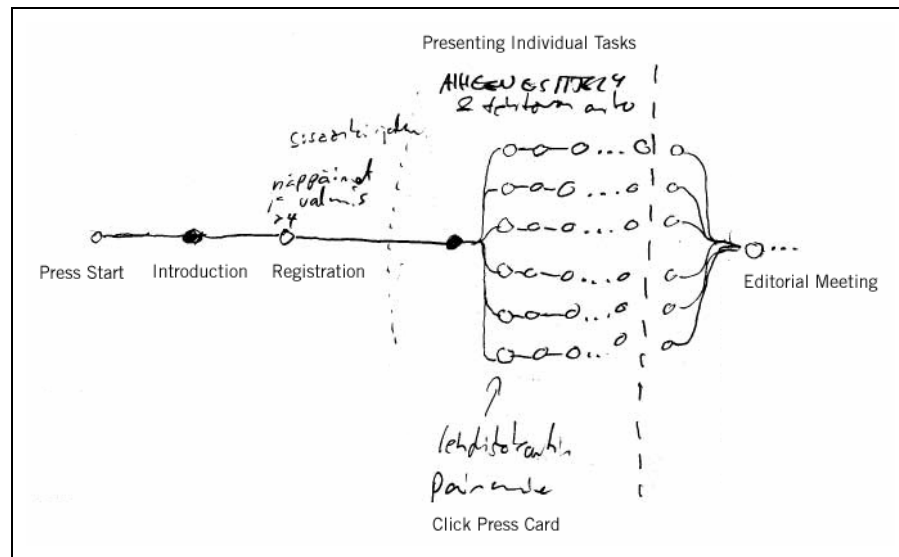


Figure 20 Individual tracks

In the individual tracks sequence students scatter around the environment to search for information and conduct personal interviews. As the name implies, in individual tracks sequence each of the six students has different task assigned to her. When the group has enough information to go forward and discuss the story all group members are called back to the news desk.

The manuscript lists all events, or steps, in detail. As an example, consider the following "task list" that describes the steps required for student 2 in the Fresh Kiss case to complete her individual tracks sequence (adapted from Expology, Sanoma Corporation & 1000&1 Digitale Eventyr, 1999 pp.14-24):

1. click press card on screen
2. go to hospital and ask a doctor about the disease (ask two patients as well)
3. check protest flyer on the table
4. call school headmaster and ask about his interests (line is busy)
5. Beaver calls in after a while
6. interview Doctor Illi at the video phone booth
7. call editor-in-chief (alternate responses depending on whether student lied about her identity in the interview or not)
8. send e-mail to Consumers Authority
9. read e-mail response
10. check archive to find any information on bleaching agents and the EU
11. call Kim Fox (student now has enough information to return to news desk)
12. Beaver calls in
13. check toothpaste advertisement on shop window
14. go to magazine stand and read Pop Magazine

Appendix 3 (p.65) contains an excerpt of the original manuscript describing the first eight tasks. Interview at video phone booth (step 6) is described in another manuscript. Clearly visible in the excerpt, each step invariably contains information or hint about the next one.

I will highlight some important details in the above sequence. Steps 1, 6, 8, and 9 involve working with graphical user interface on a touch screen. Steps 4, 5, 7, 11, and 12 are examples of placing and answering phone calls on the handset. Although not easily visible in the short description, steps 2 and 14 make use of the doorbell system; student has to find the correct doorbell button and press it to listen to appropriate character talk. As mentioned earlier, the current software implementation does not explicitly track the doorbells, so skipping one does not affect the internal state of the simulation in any way. However, skipping clues is bound to confuse the student and possibly make her wander off the correct path.

The Doctor Illi interview in step 6 has several substeps embedded within it. The interview consists of several questions that the student can choose to ask from the character. Depending on the selection, the helper character Beaver may be shown on the screen as an animation, guiding the student to right direction. Step 6 also highlights an important concept in the script: conditionality. For example at a certain stage, the student is offered an option to pose as a researcher instead of a journalist in pursuit of the scoop. Should she choose to lie about her identity, the editor-in-chief will give the student a scolding about her misbehavior in the response in step 7. To enable conditionality such as this we needed to make it possible to transfer information in the form of variables between two independent parts of the script (the video phone booth script and the individual tracks script).

Steps 3, 10, 13, and 14 are examples of steps taking place fully in the physical domain, with no connection to the information system. However, they are just as important as the other steps in advancing the story and keeping it coherent.

The student is required to change location after steps 1, 5, 7, 9, 12, and 13. This physical path is shown in the following diagram:

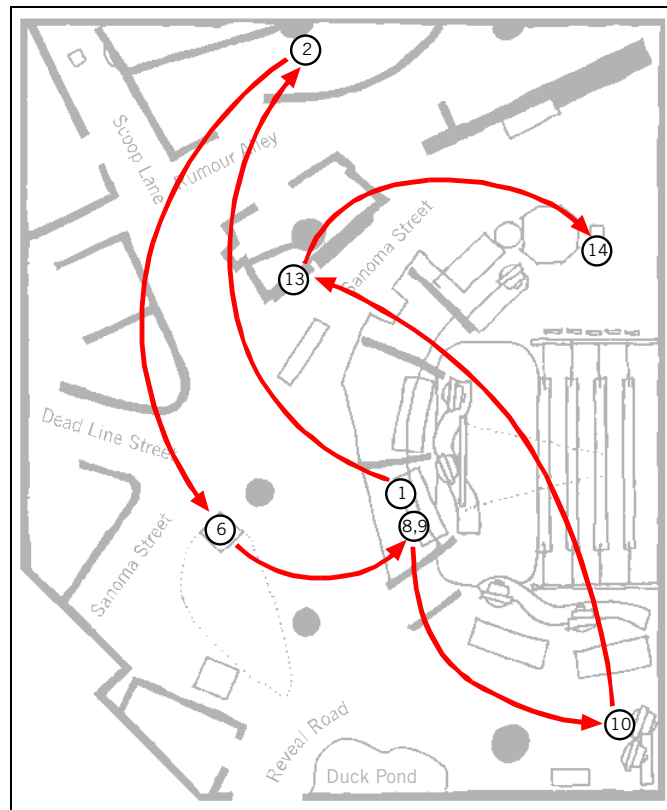


Figure 21 Student movement in the physical environment

Individual tracks sequence is the most interesting and challenging part of the story both for the students and regarding the design and implementation of the information system. The rest of the case is more straightforward, consisting mainly of question sets with multiple answers. Question alternatives are presented in written text and answers are shown by means of video, audio and animation.

### Modeling the story

The earliest thing we needed to settle on when starting to model the story was the naming of the basic unit of manuscript. These units can be identified by text starting with HS (for example HSFF\_3\_2.avi) in the excerpt in Appendix 3 (p.65). As most units describe a step that the student needs to take it could be called an "event". We had already reserved the term "event" for other purposes so we decided to avoid it to prevent confusion. Another early alternative was the non-descriptive computer science term "node". Eventually we chose to call the basic unit of the manuscript a "resource", as it usually refers to another entity, such as video or animation clip, audio file, or physical paper. A resource also contains textual description of its referee.

To pin down the structure of the story into something that we could turn into computer data we originally turned to terminology used in movies (introduced for instance in Laitinen, Raikie & Viikari, 2003). An early and quick attempt to capture the structure of a movie resulted in the following hierarchy: act (usually three acts), sequence, scene, and shot (or take).

The Internet Movie Database offers the following definitions (Internet Movie Database Inc., 2003):

- scene: continuous block of storytelling either set in a single location or following a particular character
- shot: continuous block of unedited footage from a single point of view
- take: a single continuous recorded performance of a scene.

Trying to map the above terms with the manuscript proved difficult. For example, we originally insisted that each one of the five cases would constitute one "act", and each screen (graphical user interface) would be called a "take". We quickly realized that linear narrative (movie) and interactive narrative (journalist simulator) were rather incompatible, and started working more on the terms of the latter.

A second attempt to structure the story resulted in a division to case, sequence, scene, path and node. Each *case* would consist of several *sequences* that are to be experienced one by one in a linear fashion. Each sequence in turn would optionally contain *scenes* or *paths*. These could be further broken down into *resources*. The disadvantage of this approach was that it still omitted a label for the overall simulation program: we had to be able to run the basic program (consisting of five totally different and separate cases) or the alarm program (consisting of five nearly identical cases of shorter duration).

The final hierarchical structure for the story is shown in the following diagram:

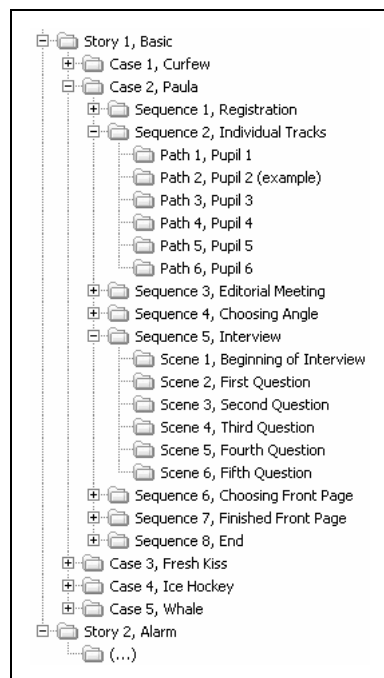


Figure 22 Story structure

As shown we ended up adding one more level of hierarchy on top of the cases: a *story*. We could now model basic and alarm programs as distinct stories. It is also worth noting that *scene* and *path* are mutually exclusive. A "normal" *sequence*, such as editorial meeting, consists of scenes that are displayed sequentially. The individual tracks sequence is special one: it consists of paths, that are run concurrently and independently of each other.

## Analyzing the manuscript

Some parts of the manuscript are reached only if certain conditions are met. Because the earlier system did not have a formal story model it was rather difficult trying to extract all these different kinds of conditions from the source code, so we decided to consult the manuscript once again. In the original manuscript conditionality and other comments are marked by different color and with two slashes (//) after the resource name and heading (see Appendix 3, p.65). I use step 4 of Fresh Kiss student 2 sequence (p.46) as an example of a conditional statement:

```
HSFF_3_2_5.wav      Telephone - Headmaster at Foxila Secondary School, 16 818
//Line is busy. Trigs HSFF_3_2_6.wav first time, same result every time but doesn't trig
```

*Table 8 Conditional statement in manuscript*

In effect, the previous condition means that the system should wait until student dials number 16818, respond with busy signal, and after a while call her back with HSFF\_3\_2\_6.wav audio file (where Beaver gives the student further instructions). Clearly visible in this example these conditions are meant for human communication only and as such are not directly suitable for computational processing. However, there being hundreds of pages of manuscript we needed a way to collect and analyze all these statements so that we could properly design our scripting engine.

We ended up implementing a text processing tool in Java that in essence reads a manuscript file exported from Microsoft Word and outputs a tab-delimited text file that is suitable for viewing in Microsoft Excel. Based on color coding and some special processing we could identify the condition clause of each resource. Obviously most conditions are used many times over, so we needed to manually edit the output file and remove any duplicates. The original list of about 250 or so conditions was narrowed down to the following:

```
after <time>
if [not] <condition>
    more/less than <num> pictures
    no answer, no call, one has pressed, one is missing
on <event>
    all finishing signal, click, fax received, finishing
    signal, mail sent, video played
    call, read mail, editorial meeting
on <choice>
    answer, response, general response
state <name> <value>
    paid = true/false
if <num> students
if <somebody> talks with student
until all but one pressed <button>
when <sequence> is over
after <Nth> student <did something>
after all students have <finishing signal>
```

*Table 9 Conditions in manuscript*

Although not a formal definition the above was a great step towards comprehending the different types of conditional situations our scripting engine was required to cope with.

## Information content

In addition to the video, audio and animation content important parts of the story are delivered as plain text. This text content is described word for word in the manuscript files and we decided not to make it separate as that would have resulted in yet another version control challenge.

As described in the previous section we already had some rudimentary tools for processing the manuscript files. What was required, in essence, was a way of extracting the text content of each resource in the manuscript and placing that into the system database. For this we developed the following process:

1. edit or update manuscript in Microsoft Word
2. export document as HTML
3. use designated tool to parse HTML file and export plain text
4. use another tool to import plain text into database.

The above process appears simple and streamlined. Before it became that easy, however, we needed to redesign the document layout of the manuscripts, update each one of them, remove any excess text and make sure each fragment of text is marked with an appropriate Word style. The HTML parsing part fully relies on text being properly marked so that each item can be placed into appropriate field in the database.

The final set of recognized styles in the manuscript is as follows:

STYLE	COLOR	DESCRIPTION
Condition	Red	Informational. Denotes a condition that has to be fulfilled before a resource is reached.
Note	Bright Green	Informational. Metadata (comment) related to a resource.
Sequence	Blue	Informational. Heading that separates larger parts of the story. Not copied to the database.
Heading	Black	Resource heading. In addition to the actual heading text it contains the unique identifier (such as HSF3_2_5.wav), separated by a comma.
Take out suggestion	Dark Yellow	Resource (part of the story) that is proposed to be removed. Not copied to the database.
Normal	Black	Description (body text) of the resource.

*Table 10 Manuscript style syntax*

Apart from the sequence headings and take out suggestions all other details are copied to the database. For text resources description (body text) of the resource is exactly what the students get to see on the touch screens.

In the process of updating information content import speed is not an issue as the manuscripts are modified rather infrequently. More important, however, is the fact that the original manuscripts remain as the data source and archival format. This means that scriptwriters can work on the scripts independently of technical personnel, and once they have finished the updated resources can be imported into the database in one lot.

## Modeling behavior: state machines

The system design so far included:

- a method of aggregating user input and delivering output through client terminals
- a mechanism for converting user input into events that are delivered within the system
- preliminary list of conditions that affect the way the story advances
- story resources and textual content available in the database

What we needed now was an exact way of translating the procedural parts of the written manuscript into computer-executable form. By looking at our previous accomplishments we identified the following key concepts: events and actions. Events always originate from user input, and actions are the tasks performed by the system based on those events. Actions usually result in output that can be experienced by the user.

Events and actions lead us to investigate state diagrams<sup>27</sup> and state-based systems modeling in general. Fowler and Scott (2000 p.119) define state diagrams as follows:

*State diagrams are a familiar technique to describe the behavior of a system. They describe all of the possible states that a particular object can get into and how the object's state changes as a result of events that reach the object.*

Important aspects in creating UML-compliant state diagrams are states, transitions, events, guards, and actions (Ibid.). A good state diagram shows clearly what transitions (possibly with prerequisites) are legitimate in any given state of an object.

One of the most common ways of implementing a state machine in a programming language is by using nested switch case statements (Martin, 1998 p.5). Whilst providing an example Martin also lists the disadvantages of this approach (Ibid. p.6):

- the code is not clear
- large state machine implementations are very difficult to read
- the logic and behavior of the state machine are bound together.

Gladly, he provides another way of describing a state machine is: a state transition table (STT). Each row of an STT lists current state, expected event, resulting state, and attached action (Ibid. p.8). This style of representation was used as a basis of our implementation.

---

<sup>27</sup> State diagrams usually describe the behaviour of a finite state machine (FSM), defined in the Aggregating user input chapter (p.33).

## Story script

We created a proprietary scripting language for controlling the story, aptly named *story script*. In the final implementation story script is stored as rows of data in the database. Different fields are described in the following table:

NAME	DESCRIPTION
Block	Script block. Each story sequence comprises a distinct script block. Script blocks may be reused. For example, the registration sequence is always the same regardless of case.
Position	Line number (current state). Having multiple lines with the same number creates an if-elseif statement.
Event Source	Name of event source (device). May be an alias.
Event Name	Name of event. May be internal class method.
Event Parameters	Optional parameters delivered along with the event itself. Analogical to guard-conditions in UML notation. All conditions must be met for the designated action to occur.
New Position	Line number to jump to when this exact event has occurred and after corresponding action has been performed (resulting state).
Action Target	Name of action target (device). May be an alias.
Action Name	Name of action. May be internal class method.
Action Parameters	Optional parameters for the action. Some parameters are obligatory for certain actions.
Resource Name	Unique identifier of a related resource. Used to refer to audiovisual and text content.

Table 11 Story script table fields

Referring back to Individual tracks structure section (p.46), the following fragment of final story script shows how steps 1 to 12 are handled by the system:

BLOCK	POSITION	EVENT SOURCE	EVENT NAME	EVENT PARAMETERS	NEW POSITION	ACTION TARGET	ACTION NAME	ACTION PARAMETERS	RESOURCE NAME
3220	1	edit	press	id=presscard, index=2	10	edit	clear	id=instruction	
3220	10				20	tvmonitor	play		HSFF_3_2.avi
3220	20	tvmonitor	played		30	edit	text	id=instruction	HSFF_3_2.txt
3220	30				35	class	enablePresscard	index=3	
3220	35				40	handset	init	number=16818, state=busy	
3220	40	handset	called	number=16818	50	script	settimer	id=p2majavasoitt aa,timeout=10	
3220	50	parser	timer	id=p2majavasoitt aa	55	handset	call		HSFF_3_2_6.wav
3220	55	handset	played		60				
3220	60	videophone	interview	resource=HSFF_3 2_7.vph, lied=1	65	handset	init	number=32772	HSFF_3_2_8a.wav
3220	60	videophone	interview	resource=HSFF_3 2_7.vph, lied=0	65	handset	init	number=32772	HSFF_3_2_8b.wav
3220	65	info	email	type=send, resource=HSFF_3 2_9_int_1.eml	67	class	emailSent		
3220	67	info	played		70	class	sendEmail		HSFF_3_2_9_int_1 re.eml
3220	70				80	info	form	id=infoemail	
3220	80	handset	called	number=18268	90	script	set	finished=1	
3220	90				100	script	settimer	id=p2majavasoitt aa2, timeout=15	
3220	100	parser	timer	id=p2majavasoitt aa2	110	handset	call		HSFF_3_2_12.wav
3220	110	handset	played		0	script	end		

Table 12 Example of final story script

A script parser essentially starts from the beginning of a script block and advances according to received events. There are multiple separate parsers running within the system concurrently, each serving different client terminal, case, or path. The above script is a good example of the capabilities of the script engine. I will highlight different aspects of the script in the following discussion.

Line (position) 1 shows how event parameters are handled. At this stage, there are a maximum of six press cards on the edit terminal screen, one for each student in the group. For all press card buttons the *id* is identical ("presscard"), but the *index* varies. As this is the individual track script for student number 2, the script only advances when the press card button with corresponding index is pressed.

Lines 10 and 20 show how media playback operations are managed. All video files are stored on client terminals and therefore they are of indeterminable length from the server's point of view. We needed to build a mechanism for notifying the server when a playback operation has finished. These notifications are delivered using *played* events.

The action of line 20 shows how text is delivered to client terminals. Depending on the active screen (form) there can be a number of text fields that may be altered by the server. These are referred to using distinct *ids*. The actual text to be shown is not written explicitly in the story script, as it would be difficult to update. Also, we did not want to mix program logic with actual content. Instead, the text fragment is referred to using its unique resource identifier. The text users get to see on the display is exactly what was carried over to the database from the actual manuscript file.

Multiple actions can be performed consequently. For example, lines 30 and 35 do not require any events to occur and therefore the corresponding actions are executed right away. Line 30 also shows another interesting feature of the script engine. We wanted to leave the script generic so it could be reused in another types of training simulations as well. With this we recognized the need for special, tailored actions that are dependent on the exact simulation being run. Therefore the script engine offers a possibility of connecting the script with actual Java class methods, on both event and action sides. On line 30, action target is "class", and action name is "enablePresscard". *enablePresscard* is actually a method in a designated class, performing multiple actions at once. Action parameters are also available for the method, allowing great flexibility within the code. A class method can also decide if it is necessary to jump to another line in the script or just keep on processing according to the current location.

Line 35 shows how the phone numbers table is altered. Using the *handset.init* action a certain phone number is set into busy state for a certain handset. After this any time the student attempts to call that number it remains busy, until it is explicitly changed again by the script.

On line 40 the system waits indefinitely until the student makes a call to number 16818 (that was initialized on the previous line). As the phone number is now busy, the student is left wondering what she should do next. Inside the system a 10-second timer is initialized on action side of line 40. The parser moves on to line 50 and waits there until this timer is expired, which triggers a call back to the student. This is exactly what happens in the description of step 5 (p.46).

Line 55 is analogous to line 20 in waiting for a long playback operation to end. There is no action to perform, and the parser just proceeds to line 60. Line 60 is special; it is actually two lines. This construction creates an if-elseif structure familiar from most programming languages. At this stage the student is supposed to conduct an interview at the videophone booth. The actual script governing the actions at the videophone is specified elsewhere. The script above, on the other hand, is only interested in the outcome of the interview; whether the student lied about her identity or not. This information is delivered from the videophone script parser to this parser using a designated event (*videophone.interview*) with a parameter called "lied". Depending on this the phone numbers table is altered accordingly, with either an encouraging or reproaching response from the editor-in-chief.

Lines 65 and 67 show how simulated e-mail traffic is managed. First the script waits until the student sends correct e-mail to "Consumers Authority". The special class method "emailSent" plays back an animation of the Beaver, and the script waits until the playback ends (line 67). After that a "reply" is posted straight away to the info terminal.

Line 80 equals step 11 (p.46). The student now has all necessary information to return and discuss the case with other students in her group. A special variable "finished" is set to notify a Java class that this student has finished her individual track. Once enough students have finished their tasks the whole group is called together and the simulation moves on to the next sequence.

As the final thing in the script the Beaver calls back the student after 15 seconds, telling her to check out some less important leads while waiting for the rest of the group to finish off with their tasks.

## EVALUATION

### Product

When discussing Piste I find defining the actual product always challenging. What we set out to do was an upgrade of the underlying software system. Content and hardware issues were deliberately left outside the scope of the project. However, the journalist simulator is an integrated whole and making significant modification to any part of it is bound to affect other parts as well. In addition to designing and implementing new software we had to edit the story, modify and cut out audiovisual material, work on user interface issues and provide documentation for guides and administrators.

In the beginning of the project we promised to deliver a "reusable platform for controlling a situated simulator environment" and "a tailored version [of that] for Piste journalist simulator" (Araqua, 2002f p.2). This tailored version is the *product* that I am evaluating here.

### Meeting the requirements

#### Piste diaries

After the initial deployment and fine-tuning period both we and the client have been satisfied with the performance of the product. As we did not have any scientific tools for measuring the "quality" of the simulation our best option of comparing the old and the new system was a manual analysis of the Piste diaries that contain informal descriptions of simulation runs written by the guides. In addition to general comments any problems encountered during simulations are also listed there.

To make analysis easier the reported issues were distributed into the following categories: reboot, hardware, printing, software, user, and sound server. Some of the difficulties cannot be addressed into any of the mentioned categories so they are listed as "other". Some problems, like a requirement to reboot, may be caused by our software, operating system, device drivers, physical hardware, or a combination of these. These are very hard to track down. We were mostly interested in distinct software problems, as these were in our power to solve.

Before introducing the new system a very rough estimate of totally successful simulations that had no problems with software or hardware whatsoever was 40% at most. That means that 6 out of 10 simulations had some glitches along the way. After developing and deploying the new system we managed to up this figure to 61% (Araqua, 2002e). If problems that cannot be helped with our software solution, such as issues with hardware, printing, users, or sound server, are disregarded the figure is increased to 81%. The remaining issues are mostly related to rebooting the clients. While disruptive, these situations are mostly recoverable under the control of the new software system.

## Requirements revisited

I will once more recall the major requirements for the new system:

NO	USER NEED OR ISSUE	REQUIREMENT FOR NEW SYSTEM
1.	Technical problems frequently affect students' work and unnecessarily shift the focus of attention to technology.	The system should be more reliable than the previous one. As a safety measure, in case of an error the system should be easily and quickly recoverable.
2.	Some local (limited to one workstation) problems are reflected to the whole simulation (all workstations).	Workstations need to operate independently of each other.
3.	Students occasionally get frustrated with the "slower" parts of the simulation. Sometimes student groups exceed the time reserved for them.	The overall duration of the simulation should be shorter. It should be possible to start a special 15-minute "alarm case" simulation separately of the basic program.
4.	It is easy for the guide to start wrong simulation by mistake. The state of the simulation cannot be monitored clearly enough. Front pages (the final outcome of the simulation) cannot be shown on the video projector.	The user interface of the guide's workstation should be redesigned. New monitoring functionality should also be added. Front pages should be viewable on guide's workstation.
5.	In case of an error or technical difficulty there is little description or help available. Error history is not available.	System errors should be reported to the administrator in detail and stored for later analysis.

*Table 13 Requirements for the new system*

The new system has been proved to be more reliable compared to the previous one. Also, the occasional need of rebooting a client computer does not affect the state of the simulation or other clients. All guides have been instructed on how to restart a client and restore it to the state it was in before rebooting. The ultimate goal is to create a system that can run nonstop for months without a single failure. Unfortunately this goal cannot be reached without fully upgrading the computer hardware and operating systems. This is a matter of a future project.

Workstations now operate independently of each other. As all of them are controlled by the server they are even interchangeable and can be replaced on the fly, provided that necessary media material is available on the replacement machine.

The manuscripts of all cases have been trimmed down and streamlined. The stories have been examined in detail and some sequences have been removed as they did not contribute to the overall goal of the journalist simulator. As a result, the simulation is 15 to 20 minutes shorter, which allows for one more group each day. Depending on the schedule of the group in question the alarm case can now be run after the main simulation, or independently.

The guide's workstation user interface was fully redesigned, and it has been highly praised by the guides. The new user interface provides some sophisticated features such as highlighting names of students who have not advanced on their individual tracks for quite some time. The guide also has more control over the simulation; for example she can jump over certain sequences if necessary. Front pages can now be shown on the video projector which makes final discussion more fruitful.

The new system also makes administrator's work easier, mostly by being more reliable. In case of any errors they are centrally reported and logged for later analysis. All in all, the new software system requires less administrative work than the previous one.

## **Client feedback**

After the deployment of the new system we received most rewarding feedback from the guides who work in the environment on a daily basis. In individual and group meetings they have provided valuable information about the behavior of the system and numerous suggestions on how to make it even better. The guides have unanimously elected the new system to be a clear improvement over the previous one, and apparently it has truly made their work easier.

We conducted the user experience study again in spring 2003 after the system had been in production use for almost one full academic year. The results echo the impressions we had. It is interesting how the discussion and comments have moved from practical details on to a higher level - suggestions and ideas of improvement mostly regard the story and actual content now. Software-related problems are much less present compared to the initial user experience study.

All in all, better reliability is the key issue, but dozens of other features and betterments are embedded throughout the environment contributing to the overall atmosphere. What the client also appreciated was our swift turnaround times in the deployment phase of the project whenever questions or problems arose.

## **Process and people**

Considering the outcome we are satisfied with our working process. The overall project management process was based on my experience of earlier projects. Still, the Piste redesign project was certainly much more demanding than the previous ones. This called for changing some practices and refining others.

We started off with vague ideas of what we should pursue and based our initial estimates on these. In retrospect we were overly optimistic in the early resourcing and time allocation. For example, the original project plan listed roughly one third of the man hours that eventually were spent on design and development of the new system. Still I think it is of paramount importance to have a written project plan. Once there is something in writing it is much easier to communicate with the client. A project plan also serves as an internal contract within the project group; everybody involved commits themselves to the plan.

Another important observation is that project plan does not need to be written in stone. Project plan, while being a tool for controlling the project as a whole, may and should evolve, although in a structured manner. Much can be learned by comparing the initial and final plans and schedules.

A team of two is tiny. In a small team it is crucial that people get along extremely well with each other. The bonds with the client also tend to grow stronger as there is much face-to-face interaction and casual meetings. This is very different compared to my experience in working as part of a larger team that operates physically far away from the client.

The whole project group had six members; four from the client and two from the supplier. During the whole duration of the project cooperation was professional and smooth. The client was always kept up-to-date on the status of the project. An important

communication tool was a weekly status meeting with at least one member present from each side.

It was fruitful comparing the fresh view of an outsider with the seasoned one of somebody who has been working with the environment since its conception. I was the outsider asking silly but important questions while Sami Pekkola did his best trying to answer them. This was a good way of finding potential places of improvement that the original designers may have ignored by oversight.

## **Lessons learned**

One remarkable decision very early in the project was to re-engineer the software implementation of the system without touching the hardware. In retrospect it would have been better to spend a few days analyzing and testing the existing hardware before ruling out any modifications. The one remaining major problem with the new system is the occasional need to reboot a client machine. This occurs mainly on client workstations equipped with rather old video playback hardware. The combination is very hard to debug: old computer, old operating system, old video card, and old device drivers. This led us to implement our video playback module in three different ways, none of which had much improvement over the others reliability-wise. Therefore the best we could do was to provide a quick way of recovering to previous state of the simulation in the event a client locks up and needs to be restarted.

Should we have tested the hardware more in the beginning we may have persuaded the client to upgrade the hardware platform before proceeding with the redesign of the software system. However, this would have added to the amount of work which was already on the limit of being achievable given the tight timeframe.

Documentation is very important. As with the project plan, putting things in writing forces one to think about them in more depth than just discussing them verbally. Documentation also serves as a future reference should any problems or disagreements arise. While it seems laborious and sometimes unnecessary at the time, writing documentation has always recompensed my efforts in the end. Documentation is a permanent extension to human memory.

One thing that only becomes apparent when project duration exceeds three months or so is the need for instruments to measure progress of the project and quality of the end product. In our case all we could rely on was the informal written descriptions of simulation runs buried within the Piste diaries to assess the quality of the earlier software system. Measuring quality and especially attempting to quantify it in some terms is extremely difficult in this type of project. Regardless, it is very important for the client as well as the supplier to have some solid figures at their disposal when the project draws to an end and its success is to be determined.

## **FUTURE DEVELOPMENT**

Considering the Piste journalist simulator the most important task is to upgrade the hardware environment in the near future for even higher reliability. This should require only minor changes in the software implementation. On a longer time span the story content has to be gradually updated and eventually replaced. Our software platform provides a good foundation for this.

It is easy to come up with technological improvements for the system. One clear candidate is the story script; while relatively easy to write the need for "line numbers" is antiquated. The syntax of the story script should be updated to more match that of a modern programming language, such as Java.

Adding new input and output capabilities to the system should prove interesting. Having the ability to control environmental lighting in the story script or get input from proximity sensors or motion trackers would greatly increase the system's "presence" in the environment. Still the importance of user experience cannot be overemphasized. Any technological advances should not be used merely as decoration or for the surprise effect, but instead they should serve the story. Ultimately, the story defines the overall user experience and hence the success of the simulation.

Creating novel content for the simulator would be the greatest challenge. Given the possibility it would be intriguing to move from separate user groups working in isolation into something that involves true competition between these groups. Designing this kind of situated interactive content is extremely demanding. Another direction would be moving from the domain of training simulation more into that of games. The physical aspects of the environment could greatly enhance the setting of, say, a contemporary adventure game. Writing a gripping storyline and adding some role-playing aspects and even dedicated clothing could make this kind of new gaming environment a huge success.

## CONCLUSION

The main questions answered by the project and this thesis are

- How to build a distributed software system that can react consistently to events generated by multiple users in different locations within a physical environment?
- How to connect these events to reactions so that a logical storyline can be conveyed using pre-recorded audiovisual material?

We created a system that aggregates user input and delivers output through multiple client terminals. All user input is converted into events that are delivered within the system. As the system is *distributed* by definition a way of coordinating actions across a number of workstations is required. Tailored client-server architecture meets this requirement. The software platform makes no assumptions about the physical location of the different user interaction points, so users can freely operate at *different locations* within the environment. To serve *multiple users* simultaneously the system is internally multitasking. I discuss the above parts of the implementation in detail in the Central Control and Aggregating user input sections in the Solution chapter.

To *convey a logical storyline* we examined the original manuscripts and created a structured model of them. To bind the incoming events with the modeled story and furthermore to create desired output we designed a special scripting language and an engine to run scripts written in that language. By writing appropriate story scripts we made the new system *react consistently* and reliably to *user-generated events*. To manage the large amount of textual and *audiovisual material* we created a process to transfer most of actual manuscript content into system database. I elaborate on this part of the implementation in the Story processing section.

The Piste journalist simulator software redesign project has so far been the greatest professional challenge I have taken part in. Bringing the project to a successful conclusion has been an enormous learning experience. Re-evaluating the whole process and documenting it in this report has taught at least as much. Hopefully the most useful results of this reflection have been communicated to you, the reader.

## APPENDIX 1 USER EXPERIENCE INQUIRY

*The original user experience inquiry and results are in Finnish.  
Results are not reproduced here.*

You can answer the questions by evaluating the journalist simulator for instance from these perspectives:

- technology (user interface, software and hardware)
- physical environment and space
- story.

1. What works poorly and what would you like to have improved?

2. What kind of control possibilities would you like the system to provide (for example starting the Alarm case separately)?

2.1. Should it be possible to speed up or slow down some parts of the story?

2.2. If yes, which parts?

3. What is irritating?

4. Case evaluations

Please evaluate each case in the light of

- instructional value
- group satisfaction
- pleasantness on the scale of 1 to 5
- other comments

4.1. Curfew

4.2. Paula

4.3. Fresh Kiss

4.4. Ice Hockey

4.5. Whale

4.6. Alarm

5. Student satisfaction

5.1. What kind of things do the students ask often?

5.2. Which tasks do the students need most guidance with?

5.3. What issues have the students been criticizing and when have they been seemingly frustrated?

5.4. What issues have generated positive feedback and when have the students been seemingly comfortable?

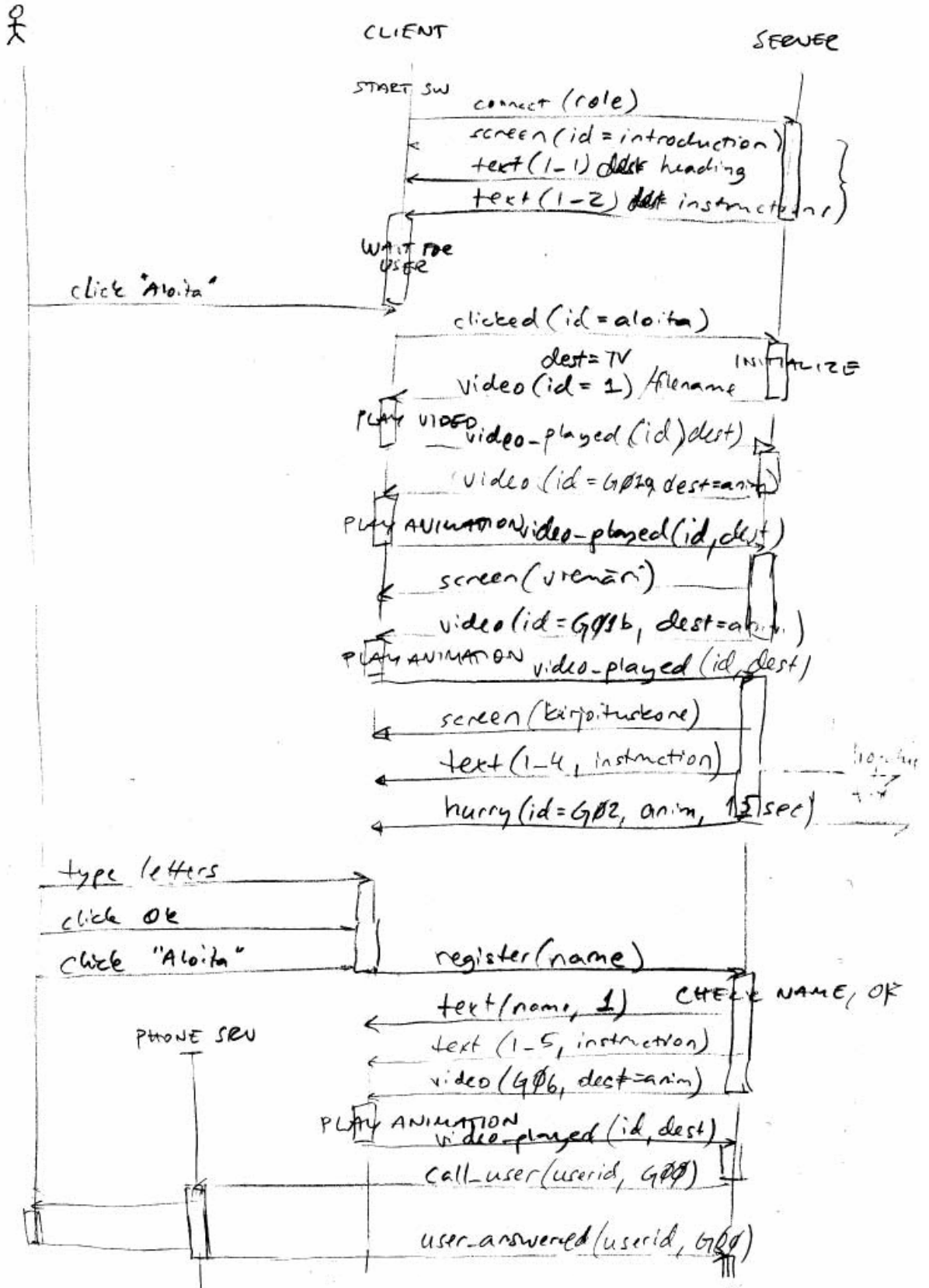
6. What works well?

## APPENDIX 2 REGISTRATION SEQUENCE

registration-seq

Y2

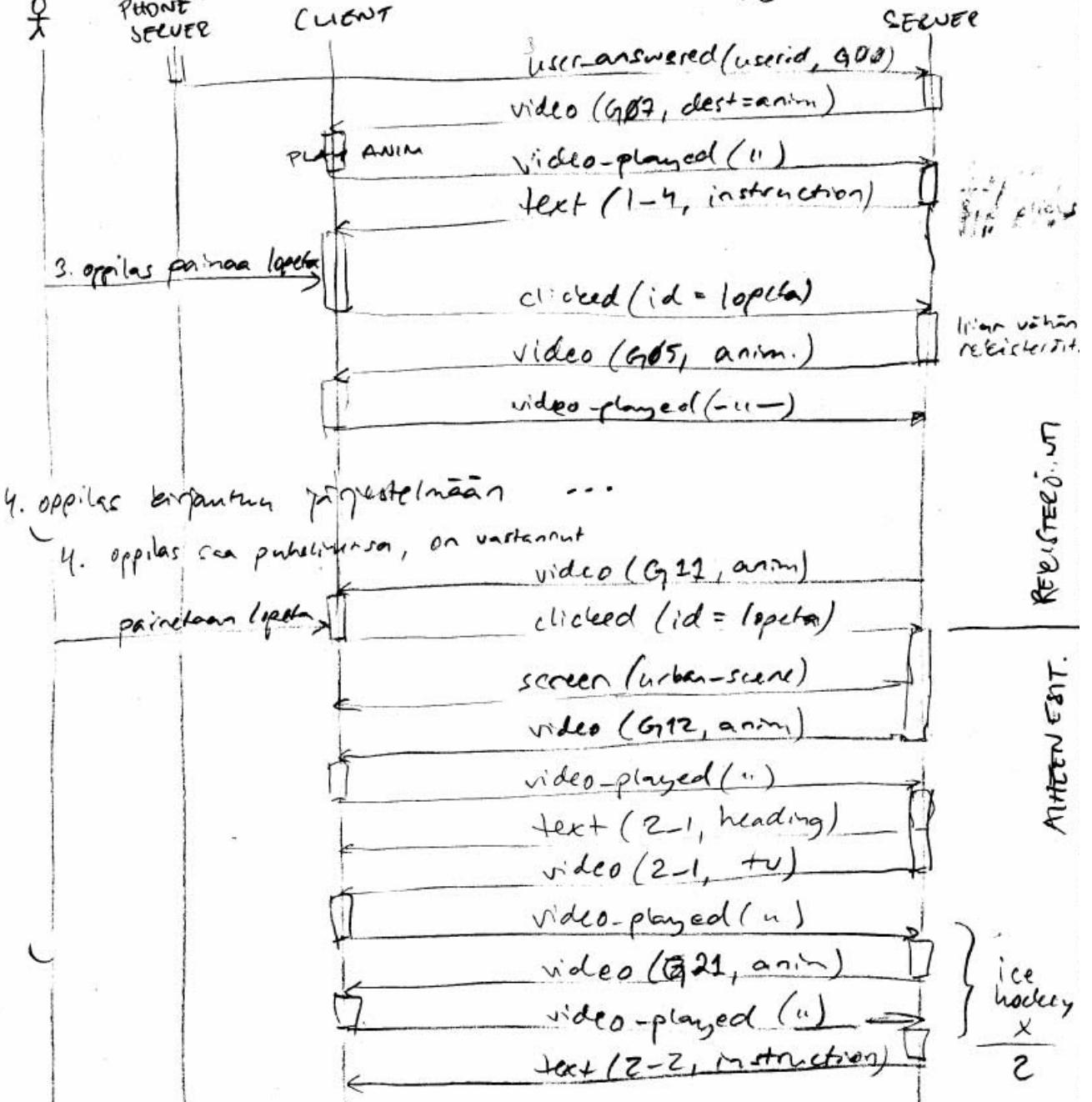
15.5.2002



registration req  
PHONE SERVER

2/2

15.5.2002



## APPENDIX 3 EXCERPT OF MANUSCRIPT

*Condensed (Expology, Sanoma Corporation & 1000&1 Digitale Eventyr, 1999 pp.14-19)*

**HSFF\_3\_2.avi** Video on TV monitor - editor-in-chief  
"We should talk to the victims of the disease. Go to Foxila hospital to see Doctor Veikko Sankila and ask about the disease. Ask him also about talking to the patients. The hospital is at 29 Rumour Alley."

**HSFF\_3\_2.txt** Text on screen - instruction  
"Go to Foxila hospital at 29 Rumour Alley to see Doctor Veikko Sankila."

**HSFF\_3\_2\_1.wav** Informant at Hospital at 29 Rumour Alley - Veikko Sankila  
*//Slightly hysterical*  
"This is a disaster! We have recorded 301 patients with the rash, and have admitted ten of them for observation. We mobilized our entire staff in the middle of the night. The symptoms of most of the patients indicate mononucleosis, although almost none of them have a fever. All the patients had tried the new toothpaste. Good God, we should get some more amoxilline quickly!! You can ask the patients about the disco the other night - for instance, Saku over there, but you are not allowed to take any pictures here."

**HSFF\_3\_2\_4.phy** Hospital at 29 Rumour Alley - Flyer on table  
"Stop exploiting the youth! The headmaster at Foxila Secondary School is risking our kids health allowing chemical experiments. Mrs. Mutru, concerned mother."  
Beaver note: "Call the headmaster on number 16 818 and ask him what his interest in this is."

**HSFF\_3\_2\_5.wav** Telephone - Headmaster at Foxila Secondary School, 16 818  
*//Line is busy. Trigs HSFF\_3\_2-6.wav first time, same result every time but doesn't trig*

**HSFF\_3\_2\_6.wav** Telephone in - Beaver  
*//Normal*  
"Howdy, your friend speaking. Great, those were good interviews, but we should have some pictures of the patients too. You can try to reach the headmaster later. You should ask the senior physician, Doctor Illi, for permission to take some pictures. Call him on telephone number 46 455 from the video telephone booth in the park at Sanomakatu."

**HSFF\_3\_2\_7.vph** Video telephone - Doctor Illi  
*//For production see script: "Video telephone". The student has to dial the phone number on the screen.//*

**HSFF\_3\_2\_8a.wav** Telephone - editor, 32 772  
*// if HSFF\_3\_2\_7\_vph\_2\_2\_1.ani in last scene.//*  
"Helsingin Sanomat, Reetta speaking..... Good you called me! I actually had a telephone call from the hospital just a minute ago. They said that a fresh journalist was there trying to get permission to take photos by pretending being a scientist. I assume that was you ..... I really don't want to hear things like that. Next time I might send you out to cover stories about a cat being stuck in a tree, or something. Anyway, we got the permission to take photos, that's good. I'll send a photographer. Listen, we just had a call from a very upset Mrs Mutru, who said that the rash was caused by the toothpaste distributed at the disco. She believes it contains some forbidden substance. Go to the internet terminal at the news desk and send an e-mail to the Consumers Authority."

**HSFF\_3\_2\_8b.wav** Telephone - editor, 32 772

// if HSFF\_3\_2\_7\_vph\_2\_2\_2.ani in last scene.//

"Helsingin Sanomat, Reetta speaking..... Great, good work! Thanks for getting us permission to take photos. I'll send a photographer. Listen, we just had a call from a very upset Mrs Mutru, who said that the rash was caused by the toothpaste distributed at the disco. She believes it contains some forbidden substance. Go to the internet terminal at the news desk and send an e-mail to the Consumers Authority."

**HSFF\_3\_2\_9.int** Internet - send e-mail, Consumer Authority

//For production see script: "Internet terminal". The student has to press "Send e-mail" on the terminal and then choose Consumer Authority from the list.//

## APPENDIX 4 PHONE SERVER LINE STATES

STATE	DESCRIPTION	POSSIBLE NEXT STATES
on hook	Phone is on hook, i.e. idle Initial state	off hook, ringing
off hook	Phone is off hook, user is dialing a number	playing, timeout
playing	User called a number, audio file is playing	endcall
ringing	System is calling the user, phone is ringing	on hook, calling
calling	System is calling the user, audio file is playing	endcall
timeout	User called a number, but main server could not provide a file name in time	endcall
endcall	End of call audio file is playing	on hook

## REFERENCES

Print

Aarseth, E. J. (1997) Cybertext: Perspectives on Ergodic Literature. Baltimore, MD, The Johns Hopkins University Press.

Bass, L., Clements, P. & Kazman, R. (1998) Software Architecture in Practice. The SEI Series in Software Engineering. Second printing. Reading, MA, Addison Wesley Longman, Inc.

Blum, B. I. (1996) Beyond Programming. New York, NY, Oxford University Press, Inc.

Fowler, M. & Scott, K. (1999) UML distilled: a brief guide to the standard object modeling language. Second Edition. 6th Printing, 2000. Addison-Wesley.

Järvinen, P. (2001) On research methods. Tampere, Opinpajan kirja.

Laurel, B. (1993) Computers As Theatre. Reprint Edition (Paperback). Addison-Wesley Publishing Co., Inc.

Microsoft ed. (1999) Microsoft Mastering: Microsoft Visual Basic 6.0 Fundamentals. Paperback. Redmond, Microsoft Press.

Murray, J. H. (1997) Hamlet on the holodeck: the future of narrative in cyberspace. Second printing, 1999. New York, NY, The Free Press.

Pressman, R.S. (1997) Software Engineering: A Practitioner's Approach. Fourth Edition, European Adaptation. Malta, Interprint Limited, The McGraw-Hill Companies, Inc.

Yin, R. K. (1994) Case study research: Design and methods. Applied Social Research Method Series, Volume 5. Second Edition. Thousand Oaks, CA, Sage Publications.

Secondary references

Buckles, M. A. (1985) Interactive Fiction: The Storygame "Adventure". Ph.D. dissertation, University of California at San Diego.

IEEE Standards Collection (1993) Software Engineering, IEEE Standard 610.12-1990, IEEE.

Available electronically

Björk, S. et al. (2001) Pirates!: Using the Physical World as a Game Board. [Internet] Proceedings of Interact 2001, IFIP TC.13 Conference on Human-Computer Interaction, July 9-13, Tokyo, Japan. PLAY Interactive Institute. Available from: <<http://civ.idc.cs.chalmers.se/publications/2001/pirates.interact.pdf>> [Accessed 7 September, 2003]

Bowers, J., Taxen, G. & Hellström, S.O. (2001) ToneTable. In: SHAPE IST Report from start-up workshops. [Internet] Deliverable D4.1. Available from: <<http://www.shape-dc.org/articles/pdf/D4.1.pdf>> [Accessed 31 August, 2003]

- Crawford, C. (1984) The Art of Computer Game Design. [Internet] Electronic version, 1997. Available from: <<http://www.mindsim.com/MindSim/Corporate/artCGD.pdf>> [Accessed 7 September, 2003]
- Granade, S. (2002) Introducing Interactive Fiction. [Internet] Available from: <<http://brasslantern.org/beginners/introif.html>> [Accessed 7 September, 2003]
- Gredler, M.E. (1996) Educational games and simulations: A technology in search of a (research) paradigm. [Internet] In D.H. Jonassen, (Ed.) Handbook of research on educational communications and technology. New York: Simon & Shuster Macmillan, pp. 521-540. Available from: <<http://www.aect.org/intranet/publications/edtech/pdf/17.pdf>> [Accessed 7 September, 2003]
- Heath, C. et al. (2001) Crafting Participation: Interaction with and around artistic, mixed media artefacts. In: SHAPE IST Report from start-up workshops. [Internet] Deliverable D4.1. Available from: <<http://www.shape-dc.org/articles/pdf/D4.1.pdf>> [Accessed 31 August, 2003]
- Marner, J. (2002) Evaluating Java for Game Development. [Internet] Department of Computer Science, University of Copenhagen, Denmark. Available from: <<http://www.rolemaker.dk/articles/evaljava/Evaluating%20Java%20for%20Game%20Development.pdf>> [Accessed 7 September, 2003]
- Martin, R.C. (1998) UML Tutorial: Finite State Machines. [Internet] C++ Report, Engineering Notebook Column. Available from: <<http://www.objectmentor.com/publications/UMLFSM.PDF>> [Accessed 7 September, 2003]
- Milgram, P. & Kishino, F. (1994) A Taxonomy of Mixed Reality Visual Displays. [Internet] IEICE Transactions on Information Systems, Vol E77-D, No.12. Available from: <[http://vered.rose.utoronto.ca/people/paul\\_dir/IEICE94/ieice.html](http://vered.rose.utoronto.ca/people/paul_dir/IEICE94/ieice.html)> [Accessed 7 September, 2003]
- Nilsen, A. G. (2000) Utvikling Av Journalistsimulator I Bergens Tidende Og Stavanger Aftenblad - En Studie Av Læringsmiljøets Designprosess. [Internet] Hovedfagsoppgave, Institutt for Informasjonsvitenskap, Universitetet i Bergen. Available from: <[http://www.hsh.no/home/agn/hf\\_pdf.pdf](http://www.hsh.no/home/agn/hf_pdf.pdf)> [Accessed 7 September, 2003]
- Raisamo, R. (1999) Multimodal Human-Computer Interaction: a constructive and empirical study. [Internet] Academic Dissertation, University of Tampere. Available from: <<http://acta.uta.fi/pdf/951-44-4702-6.pdf>> [Accessed 7 September, 2003]
- Schnädelbach, H. et al. (2001) The Augurscope: A mixed reality interface for outdoors. In Benford, S. et al.: SHAPE IST Hybrid physical-digital artefacts. [Internet] Deliverable D1.1. Available from: <<http://www.shape-dc.org/articles/pdf/D1.1.pdf%20>> [Accessed 31 August, 2003]
- Shedroff, N. (1994) Information Interaction Design: A Unified Field Theory of Design. [Internet] Available from: <<http://www.nathan.com/thoughts/unified/unified.pdf>> [Accessed 7 September, 2003]
- Venners, B. (1998) Event Generator Idiom: When and How to Make a Java Class Observable. [Internet] JavaWorld, August 1998. Available from: <<http://www.artima.com/designtechniques/eventgenP.html>> [Accessed 7 September, 2003]

## World Wide Web documents

- Aftonbladet (2002) Mediecenter [Internet] Available from <<http://mediecenter.aftonbladet.se/pres.lasso>> [Accessed 7 September, 2003]
- Everything2 (2003) Everything2. [Internet] Available from: <<http://www.everything2.com/>> [Accessed 31 August, 2003]
- Expology Burson-Marsteller (2002) Expology. [Internet] Available from: <<http://www.expology.com/>> [Accessed 1 December, 2002]
- Internet.com (2003) Webopedia. [Internet] Available from <<http://www.webopedia.com/>> [Accessed 1 September, 2003]
- Internet Movie Database Inc. (2003) Internet Movie Database. [Internet] Movie Terminology Glossary. Available from: <<http://us.imdb.com/Glossary/>> [Accessed 3 September, 2003]
- Laitinen, K., Raike A. & Viikari, T. (2003) CinemaSense. [Internet] Available from: <<http://www.mlab.uiah.fi/elokuvantaju/2001/english/english.jsp>> [Accessed 3 September, 2003]
- Merriam-Webster (2003) Merriam-Webster Dictionary. [Internet] Available from <<http://www.m-w.com/>> [Accessed 31 August, 2003]
- SHAPE (2003) SHAPE Summary. [Internet] Available from: <<http://www.shape-dc.org/summary/default.html>> [Accessed 31 August, 2003]
- Sun Microsystems (2003) The Java Tutorial. [Internet] Available from: <<http://java.sun.com/docs/books/tutorial/essential/threads/>> [Accessed 7 September, 2003]

## Project documentation

- Araqua (2002a) Pisteen uudelleensuunnittelu. Piste redesign. Presentation.
- Araqua (2002b) Piste, inventaario. Piste, inventory.
- Araqua (2002c) Piste, järjestelmäarkkitehtuuri. Piste, system architecture.
- Araqua (2002d) Piste, käyttökokemustutkimus. Piste, user experience study.
- Araqua (2002e) Piste, päiväkirja. Piste, journal.
- Araqua (2002f) Piste, sopimus. Piste, contract.
- Expology, Sanoma Corporation & 1000&1 Digitale Eventyr (1999) The Fresh Kiss Case.
- Tietovalta (1999) Piste Journalist Simulator. Brochure.

## INDEX

### Figures

Figure 1 Immersion continuum in Piste	17
Figure 2 Piste hardware environment	20
Figure 3 Project timeline, March 2002	21
Figure 4 Revised and final project timeline, July 2002	21
Figure 5 Peer to peer system architecture	25
Figure 6 Centrally controlled system architecture	26
Figure 7 Client-server setup	27
Figure 8 Beginning of registration sequence	29
Figure 9 Computational processing	33
Figure 10 Piste floor plan with all digital interaction points	34
Figure 11 Graphical user interface	36
Figure 12 Procedural vs. event-driven programming (Adapted from Microsoft, 1999)	37
Figure 13 Event sources and input aggregation	38
Figure 14 Telephone and answering machine event system	39
Figure 15 Device class and subclasses	40
Figure 16 Event class and related interfaces	41
Figure 17 Button click sequence diagram	41
Figure 18 Java AWT event delivery compared to Piste event delivery	42
Figure 19 Breakdown of case structure	45
Figure 20 Individual tracks	46
Figure 21 Student movement in the physical environment	48
Figure 22 Story structure	49

### Images

Image 1 A view into Piste journalist simulator environment	1
Image 2 Students working on the Fresh Kiss case at the news desk	5
Image 3 Stage decor, Conducting an interview at the phone booth, Discussion	7
Image 4 News Desk, Touch screen, Doorbells	9
Image 5 News desk	35
Image 6 Students working on the Fresh Kiss case at the news desk	45

### Tables

Table 1 Requirements for the new system	19
Table 2 Event messages in early communication protocol	30
Table 3 Command messages in early communication protocol	30
Table 4 Example of protocol messages	30
Table 5 Device input and output capabilities	34
Table 6 Device class and subclass descriptions	40
Table 7 Case structure and sequences	46
Table 8 Conditional statement in manuscript	50
Table 9 Conditions in manuscript	50
Table 10 Manuscript style syntax	51
Table 11 Story script table fields	53
Table 12 Example of final story script	53
Table 13 Requirements for the new system	57