SUBJECT: UbiMedia – Prototype
**Preface**

This thesis is the final work of our study Applied Computer Science at the Faculty of Business Information and ICT, University College Ghent, Belgium. It serves as documentation of the work done during our internship, which took place from March until May 2008 at Media IT, a research group of Halmstad University, Sweden.

**Acknowledgments**

We would like to thank the International Relations Office of University College Ghent and the International Office of Halmstad University who made this exchange program possible. We would also like to thank our supervisor Jesper Svensson and our mentor Sonia Vandermeersch for the advice regarding our work and thesis. Last but not least we want to thank our colleagues Jelle Bockstal and Martin Keereman for their support and feedback throughout the project and the support during the stressful days.
# Table of Contents

1. **Introduction** .......................................................................................................................... 4  
   1. Presentation of the company and their activities ......................................................... 4  
   2. Description of the assignment and the goals .............................................................. 4  
   3. Presentation of the stakeholders ............................................................................... 6  
   4. Description of the UbiMedia services .................................................................... 6  

2. **Software development** .......................................................................................................... 8  
   1. Introduction ......................................................................................................................... 8  
   1. Software process models ................................................................................................. 9  
   2. Process iteration ............................................................................................................... 10  
   3. Methodologies ............................................................................................................... 11  

3. **Technical solution** ............................................................................................................. 14  
   1. Choice of methodology ................................................................................................. 14  
   2. Requirements ............................................................................................................... 16  
   3. Prototype evolutions ...................................................................................................... 18  
   4. Tools used ....................................................................................................................... 19  

4. **Conclusion** ............................................................................................................................ 22  

5. **Bibliography** ...................................................................................................................... 24  

6. **Appendices** .......................................................................................................................... 25  
   1. Appendix A: Requirements ......................................................................................... 25  
   2. Appendix B: Evolution of the prototype .............................................................. 27
1. Introduction

1. Presentation of the company and their activities
Media IT is a research group within the Information Science, Computer and Electrical Engineering department of Halmstad University. The group consists of 6 people, some part-time and some full-time. The research group is managed by program manager Carina Ihlström Eriksson. The rest of the staff consists of PhD. students and research assistants who participate in research matters. In addition, Media IT works with students who help in projects which in return provide them with knowledge and experience.

As the name already points out the field of work is a combination of Media and IT. The research group is interested in new forms of media (e.g. electronic newspapers) which can be deployed using information technology. The most important goal of the Media IT research group is to develop new knowledge which is useful to companies and to the research community. They use new techniques and new devices and try to develop them according to the target of the projects. Different companies work in close cooperation with Media IT. This cooperation then brings forth new knowledge which benefits the sponsoring companies, Media IT and the research community. This upwards spiral gives a huge push forward to gaining new knowledge.

The Media IT group is a non-profit organization, mostly financed by national research boards. Media IT sends in applications for projects and money can for example be granted by the national research boards in Sweden, like ITEA. This way of gaining money also implies that Media IT has to work project based. Each project is different, but there’s always a connection to both Media and IT.

2. Description of the assignment and the goals
First we will describe the general outline of project, after that the specific parts of the project will be described in more detail.

1. Ubiquitous Media
The idea of ubiquitous computing as invisible computation was first articulated by Mark Weiser in 1988 at the Computer Science Lab at Xerox PARC: “Ubiquitous computing names the third wave in computing, just now beginning. First were mainframes, each shared by lots of people. Now we are in the personal computing era, person and machine staring uneasily at each other across the desktop. Next comes ubiquitous computing, or the age of calm technology, when technology recedes into the background of our lives. Alan Kay of Apple calls this "Third Paradigm" computing.”

This definition means that ubiquitous computing is not a specific technology, but more a scenario in which computers become more numerous and fade into the background, providing information to users and embedding intelligence and computing capabilities in seemingly everyday objects (OECD 2002). The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it. (Weiser 1991).

Computers are becoming increasingly small, so much so that they are now wearable, and they are increasingly able to communicate with each other autonomously. So now people are surrounded by devices that are interacting with each other and with other devices. Designs for
these new devices in these new contexts need to support people as social beings. Visibility is critical to ensure people know what is connected to what, and consistency will only be achieved once certain standards have been evolved. Similarly the devices must provide good feedback so that people do not feel that they have lost control (Benyon, Turner and Turner 2005).

An example of an emerging concept is ‘eGadgets’. Kameas defines eGadgets (extrovert - Gadgets) as everyday physical objects enhanced with sensing, acting, processing and communication abilities. Moreover, processing may lead to “intelligent” behavior, which can be manifested at various levels (Kameas, et al. 2003).

With appliances embedded in walls, jewelry and so on, human-computer interaction becomes very different. We will input through gestures - perhaps stroking an objects, perhaps waving. One of the results of this development is that we enter a world of distributed information spaces. An information space is a combination of agents, devices and information artifacts along with characteristics of media. People make use of and contribute to information spaces as they pursue their daily activities. Information spaces allow people to plan, manage and control their activities. Information spaces provide opportunities for action (Benyon, Turner and Turner 2005).

2. General project
Our internship is part of the “Designing Ubiquitous Media Services through Action Research” project. The project leader is Carina Ihlström Eriksson and the partners for the project are publishers of magazines and newspapers: Tidningsutgivarna, Aftonbladet, Expressen, Dagens Nyheter, Göteborgs-Posten, Nerikes Allehanda, Norrköpings Tidningar, Sundsvalls Tidning, Stampen AB, Sydsvenskan and Östgöta Correspondenten.

The project targets the future of newspaper and magazines. Since the fast growth of the internet in the nineties, the publishers noticed a large decrease in sales. They want to keep up with new technologies to research the possibilities of tomorrow’s newspaper. One of the goals of the Media-IT group is finding out how far readers want to go in giving up their privacy to get personalized content in return. They also question the general interest in a service like this, adoption-related issues, new media behaviors, etc.

We developed a prototype of a future application, to envision the result of the project. This prototype will also be used for further inquiries and user testing.

The prototype consists of two different parts: a bracelet with a small touch screen, the eGadget, and an application that can be run on a bigger touch screen, the information space. These bigger touch screens should be available in several public places, so users have access to them almost everywhere.

The bracelet allows you to call, SMS, check the highlights of the latest news and access various other services. When the user wearing the bracelet comes close to a screen in a public place, the bracelet will ask if he wants to connect to this screen. If so, the application on the screen boots up, personalized to the user. It’s also a possibility to have all the processing power in the bracelet. In that case, the screen is just a ‘dumb’ screen that is powered by the bracelet. The same menu from the bracelet is also available on the screen and similar functionality is available. When the user opens the news section, the intelligent database will show the news he’s most
interested in. If the user is a sports fan, the emphasis will be on sports and advertisements will, for example, show where to buy tickets for the next football match.

For such personalization to be possible, the user should reveal quite a lot of personal information. This information will not only be used to get personalized content, but also for personalized advertising. One of the tasks of Media-IT is to find out how much of their privacy users want to give up. Our prototype of this future information space should help them to visualize the future.

3. Our Project
Our task was to write a demo of the application that shows on the bigger screen, the information space. This prototype is aimed to envision the future, although we still need to work with the tools of today. Therefore our prototype should be available as a web application, but it may not look or feel like a web application. There should be a menu, an area where the different content can be viewed, and it should be possible to minimize the content to the side of the screen. For more details on what exactly our program should do, see the functional requirements. The design and content of our program was created by a design group who consisted of designers and employees of the news papers that sponsor the project.

3. Presentation of the stakeholders
The first stakeholder of our prototype is Media IT, who wants to make use of it for their research. The design group, which consists of several people from the different sponsoring newspaper companies, is also a stakeholder in a way that they have a lot to say about the requirements and that they are very closely involved in the making of the prototype.

When we look past our prototype to the final product it should become, the costumers would be the newspaper companies, the advertisers and the end-users. The newspaper companies need an application like this to preserve their income. With the decrease in sales of regular newspapers, they need to get their income another way. They depend on the advertisers for the funding of this application. A service like this should be as cheap as possible to the end-user, preferably even free. Targeted advertising is beneficial for both the advertisers and the end-users. At last the end-users who use the service everyday should feel comfortable using the application. They would have to give up quite a lot of their privacy to get these personalized services delivered to them, so the application should benefit them.

4. Description of the UbiMedia services
Media-IT started the UbiMedia project in June 2006, so there were a lot of sketches and requirements already available when we started programming the prototype. Figure 1 is one of the first sketches we made based on the explanation of the prototype by our supervisor.
There was also a low-fidelity working prototype available in HTML (Figure 2) and one in PHP. The meaning was to develop these projects in parallel and pick the best one in the end. In April Media-IT decided to abandon the other projects and to only continue with our Silverlight project, because the other programming languages were obviously not capable of handling the animations the design group asked for.
Even in these low-fidelity programs and our first sketch, the basic layout of our prototype was already available. The menu is placed in the lower right corner, containing the communication button and three visible menu items. By dragging the menu up and down, the other services become available. By dragging a menu item to the left, a window, which we will call the ContentContainer, appears. The upper right corner of the screen is preserved for the minimized versions of the ContentContainers. We will refer to this area as the LivePreviewContainer and to the minimized versions as the LivePreviews. The whole area of the screen minus the LivePreviewContainer and the menu is referred to as the DesktopPane.

## 2. Software development

### 1. Introduction

Now that we know what we have to develop, the next step is to find out how we are going to do it. Rushing into writing the code will not help us to develop quality software. We need help and a plan. A software methodology can help us with that.

In common language a methodology means the study of a method. It can provide answers to questions like: How good is a method? What is the basis of a method? But in the field of software development the term methodology has a different meaning. The meaning comprises a complete package of techniques, tools and notations (Bell 2005).
1. Software process models

In software development a process model is often also called a methodology. In this chapter we will discuss the most known and used methodologies. But before explaining the methodologies in greater detail, we will split them up in more generic process models.

According to Sommerville we can distinguish four different general models of software development (Sommerville 2001):

- **The waterfall approach**: Every activity like requirements specifications, software design, implementation, testing and so on is treated as a separate process. The previous process has to be finished before the next process can start.

- **Evolutionary development**: This process model consists of three different activities, namely: specification, development and validation. When a first version of the specification is gathered, an initial system is rapidly developed. Then the customer input will be used to produce a system which satisfies the customer’s needs. After that the system may be delivered or reimplemented using a more structured approach to produce a more robust and maintainable system.

- **Formal systems development**: This approach uses mathematical methods to transform a formal mathematical system specification to a program.

- **Reuse-based development**: This technique assumes that there are a significant number of reusable components available to integrate in the system. The development is focused on incorporating these components into a system rather than creating them from scratch.

The first two models are widely used for software development in practice. Formal systems have been successfully implemented in a number of projects, but processes based on this model are used in only a few companies. Reuse-based development is very important today to develop applications very quickly and to reduce the costs of development (Sommerville 2001).

Because our project has to be written from scratch, we won’t use the reuse-based development model. Neither will we use the formal systems development because we lack experience in working with that model.

1. The waterfall model

The waterfall model is derived from another engineering process described by Winston W. Royce in 1970. It is called after a waterfall because the processes cascade from one to another. (Sommerville 2001). This model has dominated the software development for many year because of its simplicity (Bell 2005).

In this model the software development is split up into different steps that are independent of each other. These steps are being executed in order from the first step till the last step. When a step is finished it produces a product that will be used as input for the next one. For example the implementation cannot be started before the design is completely done (Bell 2005).

There are five consecutive activities in the waterfall model (Sommerville 2001):

- **Requirements analysis and definition**: All the different requirements, goals and system constraints are gathered together with the system users. They have to be defined in great detail and serve as a system specification.
- **System and software design**: The requirements from the previous step are assigned to either hardware or software systems. This establishes an overall system architecture.

- **Implementation and unit testing**: In this stage the actual code is written. It is mainly written as separate sets of programs or program units. Unit tests verify that every unit meets its specification.

- **Integration and system testing**: The program units are integrated into the system. On completion the system will be tested to ensure that all the requirements have been met. After testing, the system is delivered to the customer.

- **Operation and maintenance**: After delivery the system is installed and put into operation. Maintenance involves correcting errors in the system and extending the system when new requirements are being discovered.

The waterfall is not a perfect approach at all. Because of its inflexibility the cost will rise significantly when the project grows larger (Sommerville 2001). The inflexibility is due to the fact that the customer only gets to see the product at the end of the development (Bell 2005). If there are miscommunications for an important requirement, the process has to start all over again.

2. **Evolutionary development**

Evolutionary development starts with an initial implementation of the system. This implementation is exposed to the users and refined by their comments. This refining will go on through many version until a adequate system has been developed. In contract to the waterfall method the activities are carried out concurrently instead of after each other (Sommerville 2001).

There are two types of evolutionary development (Sommerville 2001):

- **Exploratory development**: The objective is to work together with the costumer and explore their requirements to deliver a final system. The development starts with the parts that are understood, other part are incorporated along the way.

- **Throw-away prototyping**: The objective is to understand the requirements of the users to develop a better requirement definition. The development starts with the parts of the requirements that poorly understood.

2. **Process iteration**

All of the models mentioned above have their advantages and disadvantages. There is a need for most systems to use different models for different parts of the systems. Therefore a hybrid model must be used. We also need process iteration where steps of the process can be repeated as the requirements evolve. The design and implementation must be altered to implement the changed requirements (Sommerville 2001).

We will introduce two hybrid models which support process iteration (Sommerville 2001):

- **Incremental development**: The software specification, design and implementation are broken down into a series of increments which are developed in turn.

- **Spiral development**: The development of the system spirals outwards from an initial outline through to the final developed system.
The idea behind process iteration is that requirements are developed together with the actual software (Sommerville 2001).

In this section we will only discuss incremental development. For further reading into the spiral development, we suggest reading the book “Software Engineering” by Sommerville 2001.

1. Incremental development

This development model is a hybrid between a waterfall and an evolutionary approach (Sommerville 2001). In this model the development is split into small projects. Every iteration has its own requirements analysis, design, implementation and testing. Every iteration, the system grows incrementally during the entire duration of the project (Larman, Applying UML and Patterns 2005).

This approach has a few advantages. The customers don't have to wait to see results of the work that has already been done. It also saves more time, because implementation can start very early without the need of complete and definitive requirements. Failure of the project is also reduced because the highest priority services have to be delivered first (Sommerville 2001).

3. Methodologies

We will now discuss two widely used methodologies that use the iterative and incremental development model: Extreme Programming and Evolutionary Prototyping.

1. Extreme Programming

1. Introduction

Extreme Programming (XP) is one of the best-known methods of the agile methods, others are DSDM (Dynamic Systems Development Method), SCRUM, Crystal and FDD (Feature Driven Development).

All agile methods share common principles and these principles are bundled in the so called Agile Manifesto. These principles were created from the analysis that older methods were too big and complex. In the new methods the communication between people and individual skill are very important to develop software successfully (Bell 2005).

2. Agile Manifesto

The manifesto consist of four core statements (Bell 2005):

- Individuals and interactions over process and tools;
- Working software over comprehensive documentation;
- Customer collaboration over contract negotiation;
- Responding to change over following a plan.

Important to know about these principles is that the left part of the statements is more important than the right part. But this doesn't mean that the right part doesn’t have any value anymore. It just means that these principles give a priority to certain choices (Bell 2005).

2. XP in practice

Extreme Programming is also an attempt to develop software more quickly. It's very suitable for projects that require 3 to 10 programmers. This methodology is more based on principles than on a step-by-step approach. Jeffries (2001) defines Extreme Programming as “a discipline of
software development with values of simplicity, communication, feedback and courage (Avison and Fitzgerald 2003).

Centrally Extreme Programming recognizes that requirements can change often and they will embrace this fact instead of letting it be disruptive to the development process.

The requirements are defined in User Stories or Use Cases. These are the steps that the system needs to do for its users. User Stories also form the base of test scenarios, that’s why tests are written first in XP (Avison and Fitzgerald 2003).

These tests in XP are actually acceptance tests. The tests are written before any implementation code is written. In the beginning all tests will fail, but as the software development proceeds the tests will pass one by one. When all tests are passed successfully the implementation is considered as complete. This approach is called test-driven development (Bell 2005).

The client of the system plays a very important role in XP. According to the XP principle the client or at least a representative should always be present as a member of the development team. The client is responsible for writing the user stories and specifying the acceptance tests. Because in XP the development takes place incrementally, not all user stories can be implemented at once. So the client has to select a subset of user stories that will be implemented in the next iteration. Normally the client selects the stories that meet the most immediate business needs.

Unlike other methodologies, in XP documentation is not the most important part of software development. Delivering the actual software is the primary goal. Therefore XP is also referred as a "lightweight" methodology (Avison and Fitzgerald 2003).

3. XP Techniques

As said before XP is not a step-by-step approach but a combination of different techniques. In XP there are 12 in total. In the terminology of XP, they are called practices. The 12 practices are (Bell 2005):

1. Replan frequently: The scope of the next release should be determined very quickly. Business priorities and complex implementations play a very important part. When the reality doesn’t follow the plan anymore, the plan has to be updated.
2. Small releases: A simple system has to be put up very quickly. The next versions are released on a very short cycle.
3. Metaphor: A simple and shared story of how the system works should guide all development.
4. Maintain a simple design: There is no need for complexity in the system. If extra complexity is found, it should be removed.
5. Testing: The development can only continue if all the tests run successfully.
6. Refactoring: Restructuring the software to remove duplicate code, improve flexibility or simplify the code.
7. Pair programming: The development occurs in teams of two programmers who share one machine.
8. Collective ownership: Anyone can change any code whenever it’s needed.
9. Continuous integration: Every time a part of the implementation passes the tests, this part should be incorporated into the whole system.
10. **Avoid overwork**: Working longer than 40 hours a week is an exception. If it does occur working overtime the next week is forbidden.

11. **Involve the client**: A live user should be a member of the team. That user should be available full-time to answer questions.

12. **Coding standards**: All code is written in accordance with rules.

4. **Evolutionary Prototyping**

Prototyping is the most frequently used method to develop software when requirements are uncertain and incomplete from the start. After each evolution the prototype will be presented to the user and the requirements will be changed until the system satisfies the needs (Maciaszek 2001).

Mostly a prototype is for demonstration purposes and will be a quick and dirty implementation of the solution. It has a GUI (Graphical User Interface) that simulates the system behavior for various events. Little or no intelligence at all is built into the prototypes. The information shown on the GUI is hardcoded instead of obtained in a dynamic way from a database (Sommerville 2001).

Because of the growing customer demands, interfaces are becoming more and more complex. This complexity makes prototyping almost a must in software development. Through prototypes the feasibility and usefulness of the system can be estimated before real implementation is started. Prototyping can help to better understand the customer needs, resolving conflicting requirements and communication problems between customers and developers (Maciaszek 2001).

Evolutionary Prototyping as a method is also a part of RAD (Rapid Application Development) (L. A. Maciaszek 2001).

5. **Rapid prototyping techniques**

Prototypes need to be developed in a very quick way so that users can give feedback on it at a very early stage. In evolutionary prototyping the prototypes have to be altered quickly to incorporate the changed requirements by the user. We could use a magical tool that transforms requirements into prototypes. But sadly such a tool doesn’t exist so far. What we can use are a few techniques to speed up the prototyping process.

1. **Use a high-level language**

High-level languages have built-in functionalities which normally have to be built up from scratch in lower-level languages. Java and .NET (VB and C#) are the two most used and known high-level languages in software development. Many tools supporting these languages have a drag and drop functionality for building graphical interfaces (Bell 2005).

The downside is that the code, produced by these tools, doesn’t always follow the coding style and rules used in the development. Also the code isn’t always as efficient as code made by programmers.

2. **Reuse components**

If parts of the system are incorporated by using existing component, the development time of the prototype can be reduced. A library of reusable components and a tool to combine these into
the system are the best way to quickly develop prototypes. These components can also be used in the final system, thus reducing its development cost (Bell 2005).

3. **Use a stand-alone machine**

If the system is a network solution to be developed, a prototype running on a standalone computer is created. This will free the developer of possible performance problems or high data volume limits while networking (Bell 2005).

4. **Ignore error handling**

In many systems error handling takes almost one-half of the development time. Most error handling deals with validation of input (for example from keyboards), handling input-output device errors, exception handling software and fault tolerant software (Bell 2005).

5. **Omit features**

It is not required that all functionalities should be built in a prototype. For example authentication, logging software and security systems are usually not a priority while building prototypes. Those components are by their nature required to be high quality. And high quality means higher production cost and development time (Bell 2005).

6. **Ignore functionality**

Sometimes, prototypes are used just for showing off the interface. There is no requirement that specifies what happens when you press a certain button (Bell 2005).

3. **Technical solution**

1. **Choice of methodology**

Due to the nature of our project being a GUI prototype with requirements not set in stone, we can rule out the waterfall method.

Which of the methodologies discussed in the previous chapter should we use? Extreme Programming or Evolutionary Prototyping? At the beginning of the project we were already familiar with Extreme programming and thus it seemed the perfect way to go.

But during the project we discovered we needed something more. We were also using Evolutionary Prototyping. Due to the fact that our project is often called a GUI prototype and will only be used for demonstration purposes, it seems obvious that Evolutionary Prototyping should be used.

The truth is that we use a mix of XP and EP. This is possible because they both use a number of principles and techniques as guidelines. We will sum up the most important guidelines that we used during development.

1. **Replan frequently**

We knew from the start that the requirements were not set in stone. Every requirement was by definition still in an experimental phase. This is the kind of environment where prototyping really shines (Kendall en Kendall, Systems Analysis And Design 2002). The requirements could change, and they have, while we were still building the next version of the system. We didn’t lose
much time because through prototyping you can easily skip writing user stories and tests and head straight to the actual implementation.

2. Pair programming
The big disadvantage of pair programming is that you lose valuable time while performing very easy tasks. Tasks that can be done by one programmer should be done by one programmer. We have done pair programming only for hunting and solving difficult bugs and to write difficult or very important parts of the system.

This way we had the best of both worlds, namely dealing efficiently with the limited time and still sharing knowledge and experience when required.

3. Simple design and refactoring
In order to keep the project organized we did some refactoring to make the prototype more flexible. It was then possible to anticipate both small and radical changes to the requirements. This was sometimes in contradiction with keeping the design simple, but the delivery of the project as fast as possible was the primary goal for every partner on this project.

4. Avoid overwork
Because there were many partners involved in this project, the communication was not optimal. Our progress was sometimes stalled by lack of content and the corresponding requirements. For example when we finally got everything to fully implement a persona, we had to deliver it on the very same day. Obviously nothing goes as smooth as you want when you need to implement new functionality. This meant overtime, but luckily this didn’t occur every week.

5. Involve the client
The large quantity of partners made it almost impossible to have a live user on our team that could represent all parties. Our supervisor acted therefore as a middle man between the clients and us. Our progress could have been faster if we had access to a live user that would have the authority to make important decisions.

6. Coding standards
We distinguished two kinds of implementations, namely High-Fidelity and Low-Fidelity code (Preece and Rogers, Interaction Design 2002). High-Fidelity code can be seen as code that has been written in a reusable and quality manner. Low-Fidelity is the complete opposite, it’s quick and dirty written code because of tight deadlines. Sometimes we evolved low-fidelity into high fidelity, but in the lowest layers of the code there is no point in spending time to evolve the code to a higher quality. That time is better spent somewhere else.

7. Use a high-level language
We have used a high-level language together with tools to quickly create an interface through drag and drop or to write it manually. We will discuss this language and the tools in a next section.

8. Ignore error handling
The project doesn’t include any keyboard input which greatly reduces the amount of possible errors that can be made by the user.
9. Omit features
The project as a whole is a GUI prototype and lacks by nature any security measures or login functionality.

10. Ignore features
This technique is used both the design group and us. We used it more on a temporarily basis, while the design group used it on a permanent basis. They lack the time to create content to let us be able to simulate a fully working prototype.

2. Requirements
A requirement is something the product must do or a quality that the product must have. Conventionally, requirements are divided into two types, functional and non-functional. Functional requirements are what the system must do. Non-functional requirements are qualities that the system must have: they involve the way the functionality operates. These may be the crucial factors in the acceptability, sales or usage of a product. They cover a number of aspects of design, including image, usability, performance, maintainability, security, cultural acceptability and legal restrictions.

For both types of requirements, note that how the technology will meet the requirement is not specified. This is a later part of the design task (Benyon, Turner en Turner, Designing Interactive Systems 2005).

1. Functional requirements
The requirements can be found in appendix A. The sources for all these requirements were the design group and the people responsible at Media IT. Since the requirements changed a lot between the beginning and the end of the project, we included a first version (appendix A1) and a last version (appendix A2) of the requirements.

Some requirements were adjusted without changing the core, for example the buttons on the ContentContainers and LivePreviews were left out in favor of the use of mouse gestures to minimize, maximize or close. Using the mouse gestures is very intuitive and allows us to leave the look of a standard Windows application behind. The requirements that gave feedback about the use of the menu were changed to be less intrusive or left out because of the same reason.

Other requirements were left out because they were no priority, for example the menu that should be able to be positioned anywhere on the DesktopPane.

Christine Perfetti of UIE writes: “A persona is a profile of a typical user; it is a description of an archetypal user synthesized from a series of interviews with real people and includes a name, a social history, and a set of goals that drive the design of the product or web site. By closely adhering to the goals of a specific persona, the designers satisfy the needs of the many users who have goals similar to those of the persona.” (Perfetti 2001)

The use of personas allowed the design group to create personalized content for a specific person, which is very well suited to demonstrate how personalized content for any person could look. We think this succeeded quite well for the content and the interests. Although they tried to create a look and feel that would fit the future, we feel they failed to capture the design that modern people find attractive. The look and feel of all personas sometimes feels a bit chaotic and overwhelming in opposite to the general minimalistic trend in design. A possible reason for this
could be that the members of the design group normally work on static content for the newspapers and had problems designing for this new medium.

The three different personas we worked on are shortly introduced here. There are more thorough descriptions and scenarios available for them, but unfortunately none of those are available in English.

1. **Albin**

Albin, or “Abbe” as he is commonly called, is a 19 year old student who still lives with his parents in a small house in the suburbs. After school Abbe often hangs out with his friends in a pub at the city centre before going home to study. On Tuesday and Thursday nights he play soccer with Vaddeköping IF, periodically combined with strength training. On weekends he could go watch a game of his favorite team Hammarby, go to a party or even go to the bar or to a concert if he can afford it. Abbe has a mobile phone which he uses to SMS his friends and family and listen to music and a laptop he uses to surf his favorite sites and download movies and music.

2. **Maria**

Maria is a 51 year old Art Director for an advertising company. She lives together with her husband Bosse and their two children Kajsa and Niklas in the center of Jonköping. From time to time she grabs a coffee in a café with her friends to discuss art or politics. She also likes to travel and visit art museums all over the world, but even away from home she still wants to keep a close eye on her family.

3. **Diana**

Diana is a 33 year old single mom to a daughter of 5, Stina. She lives in a small flat in Norrköping, not too far away from her ex-husband Raoul, with whom she shares custody over their daughter. Diana works as an assistant-manager in a local clothing shop. She likes shopping, has a high interest in fashion and visits the gym at least two times a week to stay in shape. She doesn’t read any newspaper, but reads the daily news on the internet instead. She also regularly checks blog of her interest and is looking for a partner online through a dating service.

There are also some extra requirements that differ for the tree personas. These are especially designed to meet the needs of the moviemakers. In this way, they aren’t always logical for a regular user. It’s important to keep in mind that this is a prototype and not an end user application. The requirements for the personas can be found in appendix A2.

2. **User Requirements**

Because of the focus on user experience instead of usability, the list of user requirements is quite short. We will add some more user requirements after the user testing.

| **UR1** | To personalize the look and feel. Look and Feel consists of colors, menu button-shapes, window shapes, background, navigation. |
| **UR2** | What should be seen first should be placed on the diagonal from the upper left to the lower right of the screen. |
| **UR3** | Use well readable fonts. |
| **UR4** | Don’t place too many components on the screen. |
| **UR5** | Build the application on the users’ existing mental model. |
| **UR6** | Keep the learning curve of the application short. |
3. Prototype evolutions

In this section we will show the evolution of our project and describe which parts have changed. Some parts can be easily seen in a static image while other parts aren’t visible at all. We will elaborate on these changes in the commented text.

1. First version (Appendix B1)

The first version was the actual start of the project. Although there was no interaction available at all, you can clearly see how the screen is divided into different parts. In this version the screen is split up into four parts. The topmost part is preserved for feedback, when a user interacts with an element on the screen, he should get a short description of the possible functionalities of that element.

The upper right part is reserved for the LivePreviews. These show minimized version of the services that the user has opened. The lower right part is reserved for the Menu. The Menu is a representation of the bracelet that interacts with the screen. The space that is left is called the DesktopPane and will be used to display the content of the various services.

We got feedback very quickly that the feedback bar was not really wanted. The learning curve and the usability of the prototype where not the most important aspects of the project. User experience was far more important. It was based on the idea that when a user has spent enough time with the system, there was no need to educate the user anymore and the feedback would be unnecessary.

2. Second version (Appendix B2)

The second version was still not interactive, but the design incorporated more elements. The three LivePreviews are shown in their right place.

Each LivePreview has his own close and maximize button. The globe shown on each preview is a representation of what would be the final service icon. We also incorporated a darkening layer between the minimized content and the icon.

We got the feedback on this version that the icon is not really wanted or necessary. The darkening layer was under discussion, but it will be removed in later versions.

3. Third version (Appendix B3)

In this version quite a few functionalities have been added. For instance you can now interact with the system. When you click a button to open a service in the Menu, the connected ContentContainer will be displayed on the DesktopPane and the button will be disabled and grayed out. This container can be minimized to a LivePreview or closed back to the Menu. A LivePreview can also be closed to the Menu or maximized to the DesktopPane. These actions are performed by clicking the appropriate button.

When the user just clicks on a LivePreview it will minimize the current ContentContainer on the DesktopPane and maximize the content you clicked on. Basically it means swapping the content. When you click the maximize button of a LivePreview it will cascade the content on the ContentContainer that had the last focus.

The Content Containers themselves are draggable and resizeable.
4. Fourth version (Appendix B4)
This version has lost the fancy colors to see where the different elements are placed. The Content Container that is shown on the Desktop Pane is the first real content we got to implement into our prototype. This is the first version of the news service from the Albin persona.

The Menu also got an update and all services are now reachable by dragging the menu up or down. We added white arrow shaped indicators. The left indicator is the drag indicator that will light up when a user tries to interact with a service that's till unopened. Also the top or bottom indicators will light up if the menu can be dragged up or down.

To open a service, just clicking a button won’t work anymore. You have to slide the button to the left to open it.

5. Fifth version (Appendix B5)
In the final version many requirements have been altered. Almost every action requires dragging now. Closing, minimizing and maximizing have to be initiated by dragging a Content Container to the place you want it to go. For example closing a container is initiated by dragging it to the direction of the Menu.

Animations on opening, closing, minimizing and maximizing a Content Container are now incorporated. It doesn’t provide useful functionality, but it improves the user experience quite a lot.

The Menu got his final update and lost a lot of visible elements. All borders and the large indicators are removed, due to changed requirements. A small drag indicator is only shown when you try to interact with a button. An alternative technique to show the users that the menu is scrollable, has been incorporated in the form of showing a part of the next button in the menu. When no part of a button is visible in the menu, you can’t scroll it in that direction anymore.

To give a small idea of the frequently changed requirements, you can see the final version of the same news service of the same persona Albin. We have to mention that all his services were altered in this way. We spent a lot of time to incorporate the changed content.

4. Tools used

1. Silverlight platform
Microsoft Silverlight is the counterpart for the well known and widely used Adobe Flash platform. From the user’s point of view there is not much difference between installing Flash or Silverlight. Both are a plug-in for a browser. On a modern computer it takes less than 1 minute to install Silverlight.

We are using the latest version of Silverlight that is released to the public. That version is Silverlight 2 Beta 1 and it was released on 6 March 2008 (Microsoft 2008).

With the latest version we can write Silverlight applications in any .NET Framework programming language like C# or VB.NET. Due to this fact we choose to develop our prototype in the Silverlight platform rather than the Flash platform.
We didn’t had any experience in either platforms, but we already had some experience with developing in Visual Studio and C#. Also the language to describe the interface is called XAML(Extensible Application Markup Language) and is based on the XML format. We also had some experience with the XML family, so with both experiences combined we could learn Silverlight very fast. After only one week of experimenting and reading we started developing the actual prototype.

2. Expression Blend
Expression Blend version 2.5 Beta was a very powerful tool to develop Silverlight 2 applications. With this tool it was very easy to create different GUI prototypes and experiment with animations. The generated code in Blend is solely XAML code and is very readable. This enabled us to use a lot of generated XAML code in the whole project.

3. Expression Design
Expression Design 2 looks like a competitor to the almighty Adobe Photoshop but it still lacks a lot of powerful features. Nonetheless Expression Design was very useful to us, because we could create more complex GUI designs than in Blend, and export them directly to XAML code. Trying to do the same but using Photoshop instead of Blend required a lot more time.

4. Expression Encoder
The prototype uses a lot of movies, or at least it was the idea in the beginning, and the tool Expression Encoder 2 can convert almost every format into a format that is suitable to play in Silverlight applications.

5. Microsoft Visual Studio 2008
Last but not least, this tool was the core tool of our development. It enabled us to write C# code to program complex functionality of the prototype. Debugging the code was also done through Visual Studio.

5. User Testing
The user testing we did was a “quick and dirty” evaluation as described in Interaction Design (Preece and Rogers, Interaction Design 2002). “Quick and dirty” evaluations can be done at any stage and the emphasis is on fast input rather than carefully documented findings.

1. Goals and questions
In our situation, we focus on the user experience rather than on the usability of our program, so we expected the users to encounter some problems during this test. There are some things we are not allowed to change, because the client specifically insisted on some parts to work different from existing systems. Still we thought it would be useful to formulate user requirements, especially for the parts of the program we can change. We specifically wanted to know how the users adjusted to the use of the mouse gestures, if they understood the visual feedback and if the proportions of the different parts of the interface were acceptable.

2. Selection of participants
When Mats Lindqvist asked to fill in a meeting of the focus group he was working with, we decided to use the participants of the workgroup to evaluate our prototype. This way, 7 users tested our prototype, which according to Nielsen en Landauer (1993) is enough to catch at least
85% of the usability problems. The focus group had worked with the idea of the e-Gadget and the information space before and was familiar with the general concept.

3. **Method**

Before we started the user testing, we gave a short presentation of what our prototype could do. After this, the users could individually play around with the prototype on a touch screen while we observed them. Most of them tried to mimic the presentation. When they encountered any problems, we helped them. During these individual tests, we wrote down the oral comments of the participants and the issues we saw they had. After the tests, the users were asked to fill in a questionnaire. The conclusions of the testing are listed below.

1. **Menu**

   - It isn’t obvious to the user that the menu is scrollable. We used half buttons to indicate that there were more buttons available than just the ones you can see. Perhaps the users couldn’t see the half buttons very well because of the lack of contrast with the background. We could solve this by implementing a scrollbar or giving feedback with arrows up and down.
   
   - The user is familiar to clicking buttons, so the concept of dragging buttons to interact with them was new.
   
   - A grayed out button indicates that you can’t interact with it anymore. This kind of function is available in many well-spread programs. However, users still wanted to drag the button, even if it was grayed out. Users had a hard time finding the opened service in the LivePreview or on the DesktopPane. This suggests we should have an action for dragging the grayed out button. It could for example position the service on top of the DesktopPane with visual feedback of where it came from.

2. **LivePreview**

   - The amount of LivePreviews is now limited to three. A few users wanted more LivePreviews in the LivePreview, but since there are only six services available, this could be too much. With more services available, it would be possible to make the LivePreviews scrollable, as one user suggested, but that would make it even harder to keep track of the opened services. Another user suggested making the LivePreviews smaller so there would be more space to work with on the DesktopPane.

   - The mouse gestures to close or open a LivePreview are not intuitive enough. The LivePreview opened when users wanted it to close. We should make some changes to the algorithm that decides whether to close or open the LivePreview.
3. DesktopPane

- Resizing and dragging the ContentContainers around on the DesktopPane went fairly well. Even without the visual feedback, most users remembered from our demonstration how it worked.
- The feedback animation on closing and opening operations was well understood.
- The use of mouse gestures to open, close, minimize and maximize was new to most users, but had a fairly short learning curve.

The user requirements we formulated after this user testing are the following:

| UR7 | There should be more visual feedback that the menu is scrollable. Perhaps it's only needed during the first few minutes the program is used, like some kind of tutorial phase. |
| UR8 | Something should happen when you drag a grayed out button. |
| UR9 | The mouse gestures should be reliable. |

4. Conclusion

In this last part of our thesis, we would like to reflect on the work we did in the past three months.

One of the first things our supervisor Jesper Svensson mentioned when he explained the project to us, was that the requirements were not set in stone. We didn't realize yet how much frustration this would cause us. While at the beginning the idea was to have an application with easily editable content, this changed rapidly into an application with only static content and animations. In this way, the project didn't require a lot of advanced programming knowledge. It did however challenge our flexibility. Images that were carefully cut out of the Photoshop files we received and placed into our prototype, were sometimes already adjusted by the design group before we were even finished.

We were also a bit disappointed by the work of the design group. While we expected a futuristic application that resembled the look and feel of Windows Vista or Mac OSX, the images we received looked pretty outdated. We would have been more proud of our work if it looked nicer. We were a bit embarrassed when some people attending our presentations wondered if we had thought of hiring any designers.

A last problem we encountered was communication. Because we couldn't read the available documents in Swedish, all we had to go on was the oral explanation of our supervisor. Especially at the start of the prototype this resulted in some misunderstandings. Secondly there were too many persons that made the decisions. While we did have a meeting with our supervisor every week, he couldn't always answer our questions, because he had to ask the design group first. This way we had to make assumptions about what exactly we had to do, which also resulted in lost work if we didn't have it right.

All in all, this internship was very educational. Not only did we learn to handle the constant changes and did we struggle with the hierarchy for decision making, we also got a chance to see how other people looked at our work. The user testing was a very interesting experience, because it was only when we saw people struggling with our prototype, that we realized how much we could improve our it by making some small changes. No matter how many times you
read in a book how helpful a user testing can be, it’s only when you experience it yourself that you can see the true advantages of it.

Working in the Media IT lab also gave us the chance to meet people from all over the world. Most people that visited the lab had their own idea about ubiquitous computing, which broadened our view on the subject. Our prototype probably doesn’t even come close to the future ubiquitous applications that will be developed. We do hope however, that our contribution to the action research of the Media IT lab will be useful for the development of these future applications.
5. Bibliography


### 6. Appendices

#### 1. Appendix A: Requirements

| FR1 | A scrollable menu containing the available media services. Only three of them are visible at the time. |
| FR2 | A COM (Communication) button above the scrollable menu. |
| FR3 | The menu items are draggable a bit towards the Desktop Pane. And the end of the drag, the attached ContentContainer will appear on the Desktop Pane. |
| FR4 | All ContentContainers have a default size of 800x600 pixels. |
| FR5 | Menu items that are displayed in the Desktop Pane or in the LivePreview have a smaller size than regular menu buttons and a selected state. |
| FR6 | Three buttons, one for each persona, to start the prototype in full screen mode with no transparent background. |
| FR7 | The bottom of a ContentContainer contains a Buy Button and a Search Button. |
| FR8 | ContentContainers have a resize handler at the lower right. |
| FR9 | The bounds of the ContentContainers cannot be dragged off the Desktop Pane. |
| FR10 | Content from ContentContainers can be dragged to another ContentContainer. |
| FR11 | Scrolling through the menu is initiated by dragging the menu up or down. |
| FR12 | Scroll Indicators are placed above and under the scrollable menu. |
| FR13 | A Scroll Indicator must be disabled when the end of the menu is reached. |
| FR14 | A Position Indicator is placed on the right side of the scrollable menu, which reflects the position of the visible menu. |
| FR15 | A Drag Indicator is placed on the left side of the scrollable menu. |
| FR16 | The LivePreviewContainer contains a maximum of 3 Previews. |
| FR17 | A new Preview is added on top of existing Previews. When the LivePreviewContainer is full, the bottom one will be closed and sent back to the menu. |
| FR18 | The COM Preview is always at the top of the LivePreview. |
| FR19 | Every ContentContainer displays the same personalized icon in the upper right corner. At the left side of the personalized icon, there are a minimize and a close button. In the upper left corner, there are a Media Service icon and the Media Service title. |
| FR20 | The user can send a Content Container back to the Menu by clicking the close button. |
| FR21 | The user can add a Content Container to the LivePreviewContainer by clicking the minimize button. |
| FR22 | The user can send a LivePreview from the LivePreviewContainer to the Desktop Pane by clicking the maximize button. |
| FR23 | The user can send a LivePreview from LivePreviewContainer to the Menu by clicking the close button. |
| FR24 | The Menu is draggable inside the whole Desktop Pane. |
| FR25 | When the Menu is being dragged towards the middle of the Desktop Pane, the Menu starts flipping around his Y-axis. When the flip is complete, the directional functions of the Menu are inversed. |
| FR26 | When the user wants to open a new Media Service, an already opened Content Container will be sent to the LivePreviewContainer. The Content Container of the selected Media Service will be at the same spot as the previous Content Container. |
2) **Final Functional Requirements**


New requirements: FR27 and up.

| FR1 | A scrollable menu containing the available media services. Only three of them are visible at the time. |
| FR2 | A COM (Communication) button above the scrollable menu. |
| FR3 | The menu items are draggable a bit towards the Desktop Pane. And the end of the drag, the attached ContentContainer will appear on the Desktop Pane. |
| FR4 | All ContentContainers have a default size of 800x600 pixels. |
| FR5 | Menu items that are displayed in the Desktop Pane or in the LivePreview have a selected state. |
| FR6 | Three buttons, one for each persona, to start the prototype in full screen mode with no transparent background. |
| FR7 | A ContentContainer contains a Buy Button and a Search Button. (They are only enabled if the scenario requires this, see also the extra requirements per persona.) |
| FR10 | Content from ContentContainers can be dragged to another ContentContainer. (Only certain content is draggable like this, see also the extra requirements per persona.) |
| FR11 | Scrolling through the menu is possible by dragging the menu up or down. |
| FR12 | Half buttons are visible above and under the menu when it’s possible to scroll up or down. |
| FR13 | The half button is no longer visible if the end of the menu is reached. |
| FR15 | A drag indicator appears when you move over a button and disappears again when you leave the button. |
| FR16 | The LivePreviewContainer contains a maximum of 3 Previews. |
| FR17 | A new Preview is added on top of existing Previews. When the LivePreviewContainer is full, the bottom one will be closed and sent back to the menu. |
| FR18 | The COM Preview is always at the top of the LivePreview. |
| FR20 | The user can send a Content Container back to the Menu by dragging it towards the down right corner (position of the menu). |
| FR21 | The user can add a Content Container to the LivePreviewContainer by dragging it towards the upper right corner. |
| FR22 | The user can send a LivePreview from the LivePreviewContainer to the Desktop Pane by dragging it to the DesktopPane. |
| FR23 | The user can send a LivePreview from the LivePreviewContainer to the Menu by dragging it towards the Menu. |
| FR26 | When the user wants to open a new Media Service, an already opened Content Container will be sent to the LivePreviewContainer. The Content Container of the selected Media Service will be at the same spot as the previous Content Container. |
2. Appendix B: Evolution of the prototype

1) First version
2) Second version
3) Third version
4) Fourth version
Fifth version