Interaction Views in Architectures for ActionBlocks

- To each his own

Jeanette Eriksson
This thesis is submitted to the Department of Software Engineering and Computer Science at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Computer Science. The thesis is equivalent to twenty weeks of full time studies.

**Contact Information:**
Author:
Jeanette Eriksson
Address: Norrtou 300, SE - 242 93 Hörby
E-mail: jeanette.er@telia.com

External advisor:
Peter Warren
Interactive Institute AB, Space and Virtuality studio
Address: Beijerskajen 8, SE - 205 06 Malmö
Phone: +46 40-665 71 68

University advisors:
Olle Lindeberg
Yvonne Dittrich
Department of Software Engineering and Computer Science

Department of Software Engineering and Computer Science
Blekinge Institute of Technology
Box 520
SE - 372 25 Ronneby
Sweden

Internet: www.bth.se/ipd
Phone: +46 457-38 50 00
Fax: +46 457-271 25
PREFACE

This thesis is made at the Space and Virtuality studio at Interactive Institute in Malmö. I want to thank all the personnel at the Space studio for the inspiring environment and the opportunity to be a part of the studio for the past five months. A special thanks to Peter Warren, my advisor at Interactive Institute, for all the inspirational discussions and for all that I have learned during my time at the studio.

I also want to address a great thanks to my advisors at Blekinge Institute of Technology, Olle Lindeberg and Yvonne Dittrich for their support, interest and engagement in the work. Many thanks.
ABSTRACT

This master thesis is done in collaboration with Space and Virtuality studio of the Interactive Institute in Malmö. The project ActionBlocks, at the Space studio, relate to the requirements concerning hardware for ubiquitous computing. A system of intelligent building blocks is developed to be able to build functional HiFi prototypes fast. The building blocks are distributed in space and small, cheap web servers, called TINI, integrate the devices. ActionBlocks may be regarded as physical interfaces. The intention is that systems of different ActionBlocks (tag readers, digital cameras, loud-speakers, lamps, buttons etc.) may easily be constructed to support interaction with digital media in different projects. To be able to do this the ActionBlocks need to be assembled by a flexible architecture that can change when the needs alter. The goal with this thesis is to propose a concept for such an architecture. Except for the concept the thesis also contains an investigation of related architectures to explore what user aspect they have in the various projects and an implementation of a minor prototype to discover if the concept is valid in practice.

ActionBlocks consist of an intelligent (digital) part and a physical part and it is possible to discern three different approaches towards the ActionBlocks. There are:

- Physical - Action approach where the physical part and what happens in the real world is what matters.
- Physical - Computational - Action approach where both parts are integrated on equal terms.
- Computational approach where the intelligent part is most important and this view makes it possible for an ActionBlock to only contain an intelligent part.

The approaches are entertained by three different user roles: the user, the interaction designer and the programmer. The user only interacts with the physical part of the ActionBlocks and is therefore only concerned about that part. He designs the interaction with the ActionBlocks. The interaction designer assembles the ActionBlocks into a system. He configures the system and is concerned about the performance and the appearance of the ActionBlocks. Therefore he focuses on both the intelligent and the physical part. The interaction designer designs the interaction with the ActionBlocks. The programmer is the one that controls what can be done with an ActionBlock. He designs ActionBlocks. In development only the computational part is of interest because it is the only thing the programmer interacts with. The three ways to interact with ActionBlocks have an internal relationship. Development is needed to alter the possibilities to do configuration and use. The configuration forms a platform to use, because it provides new possibilities to customize it. This leads to a division into three aspects: Use, configuration and development. The partition makes it possible to focus on one aspect at a time. The three aspects have it counterparts in three different architectures: Pure Peer-to-Peer, Peer-to-Peer with distributed service and client-server architecture. The result is that the concept for an architecture for ActionBlocks is divided into three parts. One for each aspect. The concepts suggests that when the user interacts with the system the architecture is Peer-to-Peer and when the interaction designer interact with the system it is a Peer-to-Peer architecture with distributed service and when the programmer interacts with the system he can regard it as an client-server architecture.

The concluding question is if there really is a reason to adapt the architecture to different aspects. My answer is that there is always an reason to adapt the technology to the human if it is possible.

Keywords: Software architecture, user roles, interaction, ubiquitous computing.
TABLES
TABLE 1 SUMMARY OF INTERACTION FOCUS IN THE RELATED ARCHITECTURES.................. 15
TABLE 2 SUMMARY OF THE CORRESPONDENCES BETWEEN DIFFERENT CONCEPTS .............. 21
TABLE 3 CORRESPONDENCES BETWEEN USERS, RELATIONS TO ACTIONBLOCKS AND ABSTRACTION LAYERS. .............................................................................................. 22

FIGURES
FIGURE 1 THE EXHIBITION HALL .......................................................................................... 2
FIGURE 2 TINI..................................................................................................................... 5
FIGURE 3 TAG READER AND TAG.......................................................................................... 6
FIGURE 4 FROM CONVENTIONAL INPUT/OUTPUT ARCHITECTURE TO DISTRIBUTED COMMUNICATION AND MANAGEMENT (WINograd, 2000). ......................................................... 9
FIGURE 5 WEATHER ALARM (JACOBSEN & JOHANSEN, 1998)............................................ 10
FIGURE 6 LOOKUP PROCESS FOR INFORMATIVE THINGS (BARRETT & MAGLIO, 1998) ……… 12
FIGURE 7 HIVE ARCHITECTURE............................................................................................ 13
FIGURE 8 REQUEST OF SERVICE FROM CLIENT (WALDO, 1999).......................................... 14
FIGURE 9 THE COMPONENT PARTS OF ACTIONBLOCKS..................................................... 16
FIGURE 10 THREE DIFFERENT APPROACHES OF ACTIONBLOCKS.................................. 17
FIGURE 11 TYPES OF SITUATIONS, ROLES AND USER APPROACHES TO ACTIONBLOCKS 18
FIGURE 12 INTERNAL DEPENDENCES OF THE THREE USE ASPECTS ................................. 19
FIGURE 13 TAILORING IN CONTEXT OF AN ACTIONBLOCKS SYSTEM ............................... 20
FIGURE 14 THE THREE ASPECTS IN CONTEXT OF ACTIONBLOCKS ................................... 22
FIGURE 15 PURE PEER-TO-PEER ARCHITECTURE ............................................................. 23
FIGURE 16 PEER-TO-PEER ARCHITECTURE WITH DISTRIBUTED SERVICES ....................... 24
FIGURE 17 SERVER – CLIENT ARCHITECTURE .................................................................... 24
FIGURE 18 INTERACTION BETWEEN ACTIONBLOCKS IN THE ASPECT OF USE ................. 26
FIGURE 19 ACTIONBLOCKS IN THE EXHIBITION HALL ..................................................... 27
FIGURE 20 SCHEMATIC PICTURE OF HOW THE MESSAGE TRAVEL ..................................... 27
FIGURE 21 SCHEMATIC INTERFACE TO CONFIGURE A SYSTEM (ORIGINAL IDEA PETER WARREN) ............................................................................................................ 29
FIGURE 22 SYSTEM STATES IN CONTEXT OF INTERACTION ASPECTS ............................... 29
FIGURE 23 SYSTEM CONFIGURATION ............................................................................... 31
FIGURE 24 RECONNECTION PRINCIPLES ........................................................................... 31
FIGURE 25 SETUP ............................................................................................................... 32
FIGURE 26 CONFIGURATION OF UNIDENTIFIED TAGS ....................................................... 33
FIGURE 27 LOGIC STORAGE IN CONFIGURATION SERVICE ............................................... 34
FIGURE 28 LOGIC FOR THE PROJECTOR .......................................................................... 35
FIGURE 29 LOGIC FOR A MONITOR FOR SHOWING IMAGES ........................................... 35
FIGURE 30 DEVELOPMENT ............................................................................................... 35
FIGURE 31 THE WHOLE SYSTEM ...................................................................................... 37
FIGURE 32 THE ACTIONBLOCKS IN THE SOUND PLAYER .................................................. 38
FIGURE 33 MESSAGE TRANSFER ACROSS A FIREWALL FROM PEER A TO PEER B ............ 39
FIGURE 34 ONE APPLICATION AND THREE COMMUNICATION GROUPS ............................. 40
FIGURE 35 SCHEMATIC DATAFLOW WHEN THE USER PUT A TAG ON THE TAG READER ...... 41

APPENDICES
APPENDIX I TAGREADERSINGLEDATA.XML
APPENDIX II ACTIONBLOCKDATA.XML
APPENDIX III SOUNDPLAYER.XML
APPENDIX IV ACTIVEAPPLICATIONS.XML
APPENDIX V GROUPDATA.XML
1 INTRODUCTION

1.1 Background
Mark Weiser introduced ubiquitous computing in the article “The Computer for the 21st Century” in 1991. He wrote:

“The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it.”

Ubiquitous computing (Weiser, 1991) is today a hot topic. Almost every electronic device has a processor of some kind. Our homes get more and more "intelligent" and every user interacts with several computers without realizing it.

At the time of mainframes each computer could serve hundreds of users (1-100). Then came the PC and the ratio is 1-1, one computer and one user. In ubiquitous computing we talk about many users that employ many computers at the same time (100-100) (Weiser & Brown, 1996). This changes the requirements of computers, networks and applications (Weiser, 1991). This thesis concerns the two latter and is done in collaboration with the Interactive Institute in Malmö. Interactive Institute is a multidiscipline research institute in the area of IT, digital media and art. The institute is own by Stiftelsen för strategisk forskning. Interactive Institute has a near relationship to universities, companies and culture. At Malmö University the Narrativity studio and the Space studio represent the institute. The Space studio is working in a research area where the design process in combination with information technology is in focus. The research can be divided into three themes; digital amplification of action spaces, design of spaces and the everyday life in the future. The personnel of the studio include fourteen researchers and artists and several students (ten for the moment) that are working with their thesis.

The project ActionBlocks relate to the requirements concerning hardware for ubiquitous computing. A system of intelligent building blocks is developed to be able to build functional HiFi prototypes fast. The building blocks are distributed and small, cheap web servers, called TINI (section 2.1), integrate the devices. ActionBlocks are input and output devices that also can be regarded as physical interfaces. The intention is that systems of different ActionBlocks (tag readers (section 2.2.1), digital cameras, loudspeakers, lamps, buttons etc.) may easily be constructed to support interaction with digital media. A general description of ActionBlocks is presented in chapter 4. To clarify the use of ActionBlocks let me take an example. Suppose we have the following scenario:

Carl is a teacher at the Interaction Design program at Malmö University. Every year the master students have an exhibition to expose what they have done during their studies. This year Carl that works part time at the Interactive Institute wants to enrich the visitors' experiences of the exhibition by making them a part of it. His idea is that when a visitor, let us call him Jan, arrives he can choose an item that appeal to him from a basket, a nice stone, a ball etc. Then when he sees something he especially likes in the exhibition, an image, a film, a noise, a piece of music he can associate the chosen item to that object. This is possible because the item is tagged with a small electronically tag (section 2.2.1). A representation of several objects is

1 http://www.interactiveinstitute.se
2 Tiny InterNet Interface
3 Carl, Jan and Minna don't exist. It is fictive names and the scenario is realistic but has not been realized.
kept in the exhibitioner's computer and when Jan finds something he wants to store, an association may be done by putting the item at a tag reader and by choosing the representation in a simple computer interface. It may also be possible to associate the item to a spotlight if Jan likes a special part of the exhibition. At a central place in the exhibition hall Jan can put his item on a table with a tag reader hidden beneath it. Then the chosen object is exposed. If it is an image or a movie it will be exposed at the big screen at the end of the hall for all to see. If it is a sound loudspeakers will expose it. If the item has been associated to a spotlight the chosen exhibitor's corner is highlighted (Figure 1). If Jan isn't satisfied by the result he may do the procedure again. Eventually Jan leaves the item at the table for others to use.

To realize his idea Carl need several ActionBlocks; one tag reader for the central table, one tag reader for each exhibitor, one projector, a loudspeaker set and a spotlight switch. Carl also needs some software to support the application. He knows that there already exist a possibility for a user to associate different tags to different digital objects but he also need a piece of software that takes care of the exposure of the different media. Carl then asks Minna, the programmer at the studio, to construct that software. When the software is ready Carl can configure his system by fetching all the needed ActionBlocks and put them one by one at a tag reader. The ActionBlocks is then shown in an interface along with the available software. He draws lines between the different unites to put them together. When he is finished he saves the configuration and when it is time to use the system he plugs in the ActionBlocks into the network and the system is ready to be used. The purpose of the ActionBlocks project is to provide a configurable environment that can be of assistance in the design work. The ActionBlocks system should be modified to the needs that arise in different project. When we can look beyond the computer and push it in the background we can focus on other more important things like the person on the other end of the connection (Weiser, 1991) or the work to be done. ActionBlocks intend to make this possible and accordingly the ActionBlocks project is in the area of ubiquitous computing or disappearing computing were users are surrounded of several computers without having to be
1.2 Problem
In the Space studio many different projects take place. Many of the projects require IT-support within the design process. ActionBlocks may be of assistance in these situations. The presumption is that it is easy to adjust ActionBlocks to the need of different projects.

To be able to do this the ActionBlocks need to be assembled by a flexible architecture that can change when the needs alter. The architecture shall among other things make it possible to:

- Easily move ActionBlocks
- Alter the number of ActionBlocks in the system.
- Easily maintain and develop the software.

To be able to construct such architecture different possible capabilities have to be investigated, considered and evaluated. Questions that get actualized are for example:

- How will software architecture be constructed to make it possible to connect different kinds of ActionBlocks in a system?
- Shall the system be centralized or decentralized?
- How can the architecture be made flexible to make it possible to add and remove different kinds of ActionBlocks?
- How will the ActionBlocks be located?
- When a system is constructed who keeps track of the configuration?
- How can the system be connected without programming?
- How will the software architecture be constructed to facilitate further development?

1.3 Method
The work is of qualitative character and contains two parts; an investigation and a prototype implementation. The investigation consists of literature studies and database search to examine what related work and architectures exists, to utilizing the knowledge that exists in form of design principles and flexible architectures. Then a proposal for software architecture is presented. A part of the software architecture is implemented in a prototype.

1.4 Goal, Purpose and Target Groups
The goal with this thesis is to make a suggestion of software architecture that are constructed in a way that it is possible to build different configurations of ActionBlocks from existing equipment without having to modify the software. The goal is also that the architecture will be constructed in a way that if unsupported needs for other ActionBlocks arise, there will not be need for comprehensive alterations in the existing code. It will only require completion to support the new functionality for added ActionBlocks.

The purpose of the thesis is that the architecture will be implemented at Interactive Institute to be able to use ActionBlocks in different projects. This can be done without the need of big resources to fulfill the existing and arising requirements from the different projects.

The work has three target groups; the personnel at the Space studio that is expected to make use of the prototype that is developed. At this time the space studio is involved in a EU project called ATELIER\(^4\), connecting four different

---

\(^4\) Architecture and Technologies for Inspirational Learning Environments
countries and several partners\textsuperscript{5}. The ATELIER\textsuperscript{6} project is exploring, experimenting and eventually evaluating new environments for design. The project uses two classes of students as references, a class in interaction design in Malmö and a class of architecture students in Vienna. The Space studio is to provide these students with tag readers to design with and the prototype will provide the platform for the applications. The thesis may also be of interest to this target group because it may be interesting for them to explore the decisions that make a foundation to the architecture proposal.

Beyond this target group this work approaches people interesting in ubiquitous and disappearing computing along with the students at the Master program in Computer Science at Blekinge Institute of Technology.

1.5 Structure of the Thesis
This thesis is about how a flexible architecture for ActionBlocks is to be constructed. Some of the requirements and the arisen questions are introduced in section 1.2 Problem the rest of the requirements are presented in 7.1 Requirements of the Architecture. The requirements and the questions will be dealt with in chapter seven and eight.

A system of ActionBlocks may consist of devices that are not that common and in chapter two these devices are briefly presented. In the next chapter (chapter three) a selection of related architectures are presented. The architectures are investigated to explore which use aspects they focus on. In the next two chapters (chapter four and five) there is a discussion about what ActionBlocks are and how users approach ActionBlocks depending on what role they have in the interaction with ActionBlocks. The different roles relate to the ActionBlocks in different ways and this has it counterparts in different architectures. These architectures are presented in chapter six. From the previous discussions, concepts for the ActionBlocks architectures are presented in chapter seven.

This work also contains a practical part where a minor prototype is built to test if the concepts are valid in practice and the work with the prototype is introduced in chapter eight. In this chapter the implementation of the requirements is also discussed (section 8.6).

In chapter nine the proposed architecture and the result will be discussed.

\textsuperscript{5} Malmö Högskola Creative Environments, Imagination Computer Services GesmbH (Vienna), Technische Universität (Vienna), Akademie der Bildenden Künste (Vienna), University of Oulu, Consortzio Milano Ricerche and Interactive Institute Space and Virtuality studio.

\textsuperscript{6} http://atelier.k3.mah.se/home/default.asp
2 FUNDAMENTAL DEVICES

2.1 TINI
TINI\(^7\) (Tiny InterNet Interface) (Figure 2) is a small web server and is constructed with the aim to be a voice at the network for a variety of devices. Different devices like sensors, actuators and factory automation equipment are supported by TINI. TINI is a platform for system designers and software developers that construct devices that have to be able to be directly connected to the network. TINI can interact with remote systems and users through standard network applications like web browsers (Loomis, www.ibutton.com).

The platform is a set of chips and a java programmable runtime environment. The TINI contains a TCP/IP stack and a JVM (Java Virtual Machine), but the Java API is not complete. Most of the classes in java.lang, java.io, java.net and java.util are supported, but some features in the classes that support reflection and serialization is not provided for. But they will be in the near future (Loomis, www.ibutton.com).

![Figure 2 TINI](image)

2.2 Tag Reader and Tags

2.2.1 Tags

Essential to the ATELIER project is tag readers and tags. As mentioned above an ActionBlock can be almost anything, but to be able to use non-electronically objects in a computer system they have to be integrated in some way. In this project tags are attached to the item and it gets an id that can be used in the computer system.

The tags are RFID (Radio Frequency Identification) tags. A RFID tag has a small integrated circuit (IC). The tag is able to store data in the IC memory. Usually the tag has a unique numeric id. The tag can be read and written to by using a tag reader, which means that the data on the tag may be modified. RFID

\(^7\) Developed by Dallas Semiconductor, http://www.dalsemi.com
tags may have a variety of sizes and shapes and are used in many different situations. For example the chip beneath the skin of a dog is an RFID tag that identifies the dog. The anti-theft plastic tags on cloths in stores are also RFID tags (www.aimglobal.org).

A tag can be of two types; active or passive. An active tag has an internal battery and it makes the RFID tag’s range wider but it also makes it bigger, gives it a limited operational time and a higher cost (www.aimglobal.org).

A passive RFID tag collect energy from the tag reader and therefore doesn’t need a power source of its own. This means that they can be made small and for a low cost (Finkenzeller & Waddington, 1999).

The tags used in this project are passive and have a memory of approximately 256 characters. The tag might be as small as a coin of 5Skř or in a shape of a card of the size of a bankcard (Figure 3).

### 2.2.2 Tag Reader

A tag reader consists of an antenna and a transceiver with decoder. The antenna activates the tag by radio signals and can read and write data to the tag. The antenna is the conductor between the transceiver and the tag. The transceiver controls the communication in the RFID system. The antenna might be in various shapes and sizes and it can be built into the environment. Another advantage with a RFID system is that the communication between the tag and the tag reader is invisible and that tags can be read through fog, ice, paint, wood etc. where barcodes or other optical technologies don’t work. The antenna produces an electromagnetic field and if it is required it can be present all the time but if not a sensor may activate it when needed. The tag reader can emit radio waves in a range from a couple of centimeters up to 30 meters or more. The range is dependent on its power output and the radio frequency (www.aimglobal.org). When a tag enters the electromagnetic field it senses the readers signal. The tag reader decodes the data in the tag’s silicon chip (integrated circuit) and the data is forwarded to a computer that processes it.

A tag reader is an ActionBlock (in combination with a TINI). The tag readers (Figure 3) used at Interactive Institute has a range up to ten centimeters. The smallest a tag reader in the project can get is as small as a tag and it can’t get bigger than approximately the size of an A4 paper.

![Figure 3 Tag reader and tag](image-url)
3 ARCHITECTURES FOR SMART THINGS

3.1 Related Architectures

As mentioned earlier in section 1.1 Background ubiquitous or disappearing computing is highly interesting today and so are sensors and actuators. Many researchers experiment with different kinds of sensors and environments and they suggest various approaches towards the infrastructure.

In the scenario in section 1.1 Background three different persons interact with the ActionBlocks system, Jan, the user, Carl, the interaction designer and Minna the developer. The three has different concerns about the system. Jan wants the system to be easy to use but it also has to be interesting to use it. It shall enrich his life. Carl wants the system to be flexible so that he easily may alter it if he wants to change the interaction with the system. Minna really wants the system to be easy to maintain and develop. The three roles may be contained in one person but to simplify the example the roles are represented by three different persons.

Further on some examples of related architectures are presented. Some of the architectures described are more related to the scenario than others. It may be interesting to explore what use aspects the researchers implicitly have used to implement the related architectures. Section 3.1.1 - 3.1.10 begins with a presentation of the ten different architectures and ends with small summaries of which user roles that are supported in the projects.

The collection of projects is at no means complete it is just a selection to exemplify the range of architectures.

3.1.1 Multiple Trivialities

In the Space studio another project, Multiple Trivialities\(^8\), has used ActionBlocks to achieve a tagged environment. The project is an artist project and is in different places at the same time, in an atelier in Boston, on a web page and in a gallery in Malmö. The web page is a peeking hole into the atelier to be able to follow the work. In the gallery the guests may move items (books) that are tagged and in this way alter the appearance of the web page.

The application\(^9\) has client-server architecture and consists of several parts; two tag readers in Boston, two tag readers in Malmö, logic placed at a server at Interactive Institute, a client in the gallery (applet) and a client in Boston (applet). The server handles all the communications between the clients because the logic is placed there and the logic combines the different inputs from the four tag readers to two outputs.

The user play around with the books and may in that way interact with the artist in a chat or look at her in her studio or look at her digital archive. The artist is concerned about the users experience when the system is used. The artist is like Carl in the scenario above. The difference is that she doesn't configure the system. If she wants to do something different she needs assistance.

In the frame of this project a network protocol has been developed to, among other thing, make it easy to substitute an ActionBlock to another.

The project has focused on the use and to facilitate further development. To make a comparison with the scenario the user (Jan) and programmer (Minna) roles are well provides for but it is not that good for the interaction designer (Carl).

3.1.2 Web Presence

Web Presence (Kindberg et.al; 2000) is a project performed at Hewlett Packard Laboratories. They have been exploring how to bridge the physical world and the

\(^8\) [http://ipp.interactiveinstitute.se/multipletrivialities/index.htm](http://ipp.interactiveinstitute.se/multipletrivialities/index.htm)

\(^9\)
World Wide Web for people, places and things. The bridge embraces the ability to interact with different devices through a user interface in a browser. People, places and things are provided with a web resource that can be used to store information about them. The idea is to support mobile users with their everyday activities. When a person arrives at a new place he can locate a printer to print his working schedule, or if he wants to contact a person he can visit the persons web page and make contact. The web page is updated with the location of the person and if the person is near a phone that phone will ring. If it is more convenient to send e-mail that will be done. A place may also have web presence and when a person arrives in a new town the showplaces of the town is displayed.

It is a rather large project. They use different kinds of architectures depending on what is most convenient in the different areas. An electric device may contain a web server that act on its own behalf, but there could also be a central web server that acts as a gateway to the devices. For non-electric objects as places, persons or paintings there has to be an external server that handles the web presence. The infrastructure contains both wired and wireless devices. The architecture is designed to make it easy for the user to move around and connect to things and places nearby. To discover URL’s the client device access the network and participate in a protocol for discovery. The web presence device also participates in the discovery protocol. The protocol then provides the client with the requested URL. The mobile device sends multicast to find a “service discovery” that is a meeting place for network services. The result is that the mobile device is connected to for example a printer (Kindberg et.al; 2000).

The principle of use is easy, but it provides a lot of opportunities to make choices that isn’t that easy to make. It requires some knowledge of what can be achieved by the system and the services may be of various kinds. The web presence moves across networks and it is up to the user to choose and customize the services. The project supports mobile users, but to take advantage of the opportunities a user has to have special competence. The user has to customize his environment to support his needs. The customization is of bigger importance than the use. The use really is to be able to customize or configure the environment. Whether the development is especially provided for is hard to say, but the web presence involves several networks, several kinds of devices and ways to reach them. Programmers’ work might be rather difficult.

3.1.3 Appliance Data Service (ADS)
The Appliance Data Service project (Huang et.al; 2001) explores an idea that would make users move data seamlessly among various devices. They work after three principles: bring devices to the forefront, minimize the number of device features, and place functionality in the network infrastructure. They have identified a usability problem with digital devices. The problem is that the devices often have a variety of features that the user doesn’t comprehend. There are more buttons to push and there are more adjustments to make. Another problem is that the user often has to interact with a PC to make use of the digital device. The user has to install and configure the software that comes with the digital device. Then there is the learning phase for both the digital device and the software. The obstacles are many. Huang et.al. is of the opinion that the appliances that people use should be adequate to let the user focus on the task. The key issue is to keep the devices as simple as possible without losing the usability. The device should support at least the task that the non-digital counterpart can achieve without interacting with a PC. Placing the software in the infrastructure instead of in the device solves that problem. The architecture is centralized to obtain easy update and administration.

The case study that is performed is about digital cameras and the retrieval and publishing of photos. The architecture is intended to support this type of use. A scenario may look like this: A person is on a travel and she takes photos with her
digital camera. When the camera is full she goes into an Internet café that has an ADS access point, an IR sensor. She inserts her smart card, her PIN code and selects the “photo album” application. When all three are aggregated the photos is unloaded through the IR port on the camera. The selection of “photo album” application indicates that a specific XML template is to be selected. This template determines what services are going to start. In this case it means that the pictures get scaled at a certain amount and the pictures are transferred with FTP to a specified site (Huang et.al; 2001).

The project focuses on how to make digital devices easy to use. The user can easily manage to use the system and he has some limited possibilities to customize his use. The camera only contains a limited amount of software. The severe computation lies on a server that support the users work with the camera. This means that the developer’s work is made a little bit easier by the fact that all software is at one place.

3.1.4 Interaction Spaces
At Stanford University several research groups are exploring how to develop architectures that focus on multiple users in an interaction space instead of systems as a network of different devices (Winograd, 2000). At Stanford there is a room, a workspace, containing large high-resolution displays, PDAs, tablet computers, laser pointer, cameras, microphones, floor pressure sensors etc. The workspace is designed to support several users that work together and who moves from device to device. The following scenario is an example of how a person can interact with the display. The person put his index fingers on an image on the display and slides them apart. The image gets bigger. When the image has the right size he stops. If he draws a circle around two of the images, they are selected. Then the person says, “Hold for the product page.” and the selected images can later be retrieved under the category “product page”.

To be able to use different devices spontaneously the research group has developed an architecture where the drivers are substituted with diverse observers and the server that keeps information about the system and synchronizes activities is divided into several managers (Figure 4). A blackboard, heap, is used to avoid creating explicit communication paths between devices. A context memory, a persistent storage to preserve context models is also used. Examples of context models are current context (who is were at the moment), application context (task-specific vocabularies and grammar) and personal context (speech and handwriting).

![Figure 4 From conventional input/output architecture to distributed communication and management (Winograd, 2000).](image)

After a device has produced a signal, an observer produces a specific interpretation of the phenomena that produced the signal. The phenomena triggers an event posted to the blackboard. The blackboard (Event Heap) makes it possible to avoid creating explicit communication paths to different observers. The observer can subscribe to a specific event posted by a particular device and when the event
happens the observer react on an event and produces an interpretation and the manager synchronizes the activities and then the task is performed.

The centralized blackboard procedure makes an extra “stop” in the process, but it is motivated with the increasing performance of computers (Winograd, 2000).

What roles are most supported? When the user is using the system he can move between several devices and use them as he pleases. He can focus entirely on the working task not the tools. The developer, most likely by an interaction designer’s request, predefines the possible action the user can make. The user's and the developer's task are well supported while the software components are separated and there is a shared device were contextual information is stored.

3.1.5 Tacoma

At the university of Tromsø an architecture with cellular phone and a remote server, based on a mobile code system is developed (Jacobsen & Johansen, 1998). The investigation was concerned about how to retrieve remote network services from thin clients. Most cellular phones don’t have software support to work on information from remote services. The only connection to outside services is through SMS messages. The research project has explored how to combine SMS messages with middleware system for mobile code. The Tacoma system (Johansen et.al, 1994) is used in the architecture. The mobile code can travel between sites that support the Tacoma system, have a dedicated Tacoma server. In such a system the mobile agent stores the data in briefcases. In a meet the mobile agent that takes the initiative can specify a target site and a target agent. The initiating agent may then execute code at the target site or apply for a service. In this project SMS is used to move text-message agent programs between cellular phones and the Internet. The SMS message is the transport media. Small agent programs can move from a cellular phone to a Tacoma server and back again.

As a case study a weather alarm system is developed. The first step in using the weather alarm system is to create a weather alarm on a cellular phone. The alarms are written in a high-level boolean language, not as verbose as Java or C. The SMS is sent to a service provider that sends it further as e-mail. The e-mail is inserted in a queue at the server and expanded into the Tacoma briefcase format. The weather alarm is then periodically executed at the server. If the data from the sensors triggers an alarm a notification is sent to the phone as an SMS or as e-mail to the specified address (Figure 5). The user may then inquire further information about the cause of the alarm (Jacobsen & Johansen, 1998).

In this concept the user may design his own weather alarms. They program the agents in a simplified programming language, but it isn’t that the project focuses on the programming aspect. Instead the programming the user does is a form of configuration. The user writes instructions that customizes his use of the weather alarm. After configuration the system may be used. If the user isn’t satisfied with the customization he can make the procedure again. Here we really can talk about a centralized system. All the software is at one place. The phone produces an ordinary SMS message it's all it does. Then it is up to the server to set up the alarm. All though the system is centralized and the developers task may be supported the
programming aspect is not very important in the context. It is the possibility to configure an alarm that matters.

3.1.6 **RFID**

Chef

In the Distributed System Group at the Swiss Federal Institute of Technology they have examined the requirements for an infrastructure for smart things. In the research a case study has been performed. During the case study they have implemented RFID Chef (Langheinrich et.al, 2000) that take place in an ordinary kitchen with a tag reader in the counter. All groceries are tagged. Instead of bar codes all grocery get a tag at the store. When someone is going to cook a meal, he puts some ingredients at the counter and the system lookup all recipes that match the ingredients. The recipes are shown. If there are some ingredients missing a red bar is shown at the screen. The smaller the filling is, the closer to a complete match. If all ingredients match, the bar is green. The recipes are graded after the grade of match. The green recipes are placed first and then the least red and so on.

The project use events to attach the real world with the virtual. Changes in the state of a real world thing generate an event and is signaled to the counterpart of the virtual world and the state change is simulated. Inputs from sensors are combined to form context events that model the real world and then they are modeled in the virtual world. The prototype RFID Chef is only implementing a part of the suggested infrastructure. Therefore the prototype is not a part of Internet, but the intention is that it should be (Langheinrich et.al, 2000).

The configuration aspect is not regarded in this project. It focuses on how to construct an infrastructure that supports interaction with physical objects. It tries to make it easy for the user to use and to make everyday life easier. It provides the user with a more flexible tool than a cooking book. But there is yet no opportunity to compose or add own recipes and the fact that the possibility to share recipes on the Internet is not supported indicate that the interaction designer's role is regarded as not so important at this time.

3.1.7 **Informative Things**

Barret and Maglio (Barrett & Maglio, 1998) describes a "floppy-like" interface in the article "Informative Things: How to attach information to the real world." Actually the intention isn’t to suggest a replacement for a GUI but to connect the virtual and the real world when it is suitable. The approach gives the user an impression of storing information in physical items. The storing doesn’t take place in the physical thing but rather in the network. The physical item is just a pointer to the information. The approach hides the “storing” mechanism to the user and in that way generates some advantages. The user doesn’t have to find a storing location, naming the information in a global way, select transport or set the access permissions. The physical item is uniquely recognized, which make these actions unnecessary.

The implementation contains a client, a server and a physical thing. The physical thing may be of different kinds but the floppy disk has been selected because it doesn’t need additional hardware. When inserting the floppy disk in the client, the client look up the thing id in a database and the client get knowledge of which URL the floppy disk is associated to (Figure 6). The thing is mapped to the things URL and this URL can be used to find a description of the thing. The thing’s description may contain a list of URLs and those URL point to the data that the thing “holds”. The thing description is an XML document. The service that look up the URL that match the thing id may be centralized on a well-known server or it can be distributed like DNS systems. It’s notable to recognize that the application is made in a way that only permits the owner of the Informative Thing

---

10 Radio Frequency Identification
to alter the files associated to it. This makes it unnecessary to give access permissions to the file (Barrett & Maglio, 1998).

Barrett and Maglio are pointing out that the concept can be used in many different situations, as an example it can be used when a user works with two different computers. It can be of difficulty to keep the files synchronized when doing so. An Informative Thing and a personal area network (PAN) make synchronizing files easy. The user has a PAN card in his pocket or his wallet and when he sits in front of the computer the receiver picks up the PAN card’s id. The computer displays the card icon and attached data and the user can start to use the files. When leaving the computer the icon disappear from the screen and when the user sits down in front of the second computer the icon is shown there and the user may work on the same file. The file is actually stored at a web server but the user has the impression of carrying the data with him (Barrett & Maglio, 1998).

The virtual floppies are constructed with the end-user as a target group. The project doesn’t deal with maintenance or further development and the user is not supposed to configure the system. The system shall be easy to use and as automatic as possible.

3.1.8 Invisible Interfaces

There have been many research groups exploring how to bring the physical and the virtual world together. At Xerox Park, as well as in other research centers, efforts has been made to support everyday tools and objects and add computation to them to support spontaneous interaction by using natural maneuvers and associations (Want et.al, 1999). Want et.al. have focused on how to combine RFID tags and tag readers, RF networking, infrared signals and mobile computing. Essential to the system is tags (only readable) that are attached to physical items and tag readers that are plugged in the serial port at a handheld computer to reduce the number of devices to carry around.

The research group has made several prototypes to explore the functionality. In one of the applications a tag is embedded in a wristwatch. When the watch is brought close to the tag reader the calendar program is shown at the computer. The program is adjusted to the particular user and the current time. The user can hold the computer in a normal way without activating the calendar program. It has to be a deliberate act to start the program. The wristwatch is from the start associated to scheduling and the connection to the calendar program is a good example of how to bring the physical and virtual world together in a natural way (Want et.al, 1999).

When a tag is read the tag is looked up in a semantic database. The database is a shared network database that maps the tag id to its virtual association. The descriptions are generic to support additional documents in a portable way. The approach ensures consistency to multiple users or computers (Want et.al, 1999).

---

11 Radio Frequency
The invisible interfaces focus on easy use, the interaction with the interface is preferable intuitive. The use is rather static. The wristwatch activates the calendar program. There is no possibility to customize that behavior if a desire to make the wristwatch do something else arises. The user may not be bothered with configuration and the programming aspect is also considered of minor importance.

3.1.9 Hive

Hive is a distributed agent platform that deals with Things That Think. It is concerned about ubiquitous computing. The goal of Things That Think project (Minar et. al, 1999) is to put computations and communications in everyday items, in kitchen, shoes etc. Hive is built exclusively of Peer-to-Peer interactions between agents.

Hive is built in Java and makes use of serialization and Java Remote Method Invocation (RMI) to accomplish mobility and to ensure communications between agents.

Hive is composed of three components: cells, shadows and agents. The cell may be compared with a web server. It is a program running on a computer with a known network address. A shadow is a resource for example a digital camera. A cell contains a set of shadows. The cell also hosts agents. The agents use the shadows and they communicate with each other (Figure 7). Each agent has its own location and they use a variety of local resources. The agents in Hive are building blocks in the distributed system (Minar et.al, 1999).

Hive also offers a graphical user interface to the distributed system for information and control. The computer display is another shadow and an agent can draw on the screen by sending graphical components to the shadow. Through the interface new agents can be formed from drop-down menus. Drawing lines between the agents can connect them. It’s also possible to kill or move an agent through the interface (Minar et.al, 1999).

An example of how Hive works is the use of a digital camera. A cell has a camera plugged in. The shadow of the camera supplies the hardware with a software interface. Methods in the interface may be for example takePictures() or setBrightness(). When an agent wants to use the camera it moves to the camera’s cell and makes a request for the camera shadow. Then a method call invokes the shadow (Minar et.al, 1999).

In Hive the use aspect is important. It is of great concern to make it easy for the user to join the network and it is essential that the system is robust. The goal with Hive is to preserve a network that supports everyday object. In this lies the idea to let the network fade back into the background. It is also of importance that the user can form new agents and connect them when it's needed.

3.1.10 JINI™

JINI deal with questions like scale, component integration and ad-hoc networking (www.jini.org). The JINI technology presumes that the network continuously changes both the devices at the network and how they interact. JINI makes it possible for anything with a processor, memory and network connection to offer services or use them. To upgrade the network the upgrades or updates is installed on a component and then its ready to be used. JINI is built upon Java technology
and use Remote Method Invocation (RMI) and the idea of proxy is essential. A proxy is a local object that represents a remote object (Waldo, 1999).

JINI in work means that a network service discovers a JINI lookup service and sends the service proxy to the lookup service. The client that needs a service also discovers the lookup service and sends a request to find the wanted service (Figure 8). Then the lookup service sends the registered proxy to the client and the client can interact directly with the service via the proxy (www.jini.org).

![Diagram of JINI client and service](image)

**Figure 8 Request of service from client** (Waldo, 1999)

The JINI technology may for example be used at a group of high-rise office buildings. The buildings are connected to a JINI network. A variety of services can be registered with a lookup service. The services can be services for lights, phones, elevators etc. The services can be set up in a hierarchical lookup service to support, owners, tenants, managers etc. (www.jini.org).

I consider JINI to focus on the configuration aspect. It is most important that the user can plug in a device into the network to make the network function by itself. It isn’t when the user actually prints a document at a JINI device that is in focus but how the network is put together.

### 3.2 Summing Up

The related architectures suit the three roles in the scenario in various ways. Not many would consider the programmer Minna, in the scenario, as a user but she has a relation to the system when she maintain or develop it further. She uses the architecture as a base when she expands the system and when she builds new ActionBlocks. She doesn't build from nothing. She uses existing components as platforms for new ActionBlocks. It might be considered use. Carl the interaction designer also has an ambivalent role that may not be considered as use at all time, but he too has a relation to the system. Jan, the user is the end user we normally think of as the user. How well are these three roles supported in the related architectures? Most of the projects have the kind of use that Jan represents as a primary goal. The systems flexibility has lower priority. It's most important to make the system easy to use. Some of the projects, "Web presence", "Tacoma", JINI and in some range also Hive is focusing on the configuration. In these cases it is important that a person like Carl is able to customize the systems for his needs. It shall be remembered that in these cases Jan's role is merged with Carl's. In other words the roles of Carl and Jan are considered as one. Carl's part of the merge is the bigger part. An overwhelming part of the related projects has implemented a centralized architecture. In some of the projects it is natural. In "Appliance data Service" and "Tacoma" thin clients are used and therefore a server that make the severe computation is desirable. Also "Informative things" are well supported by a client-server architecture because of the stored data is going to be reached from different computers. But in other cases like the RFID Chef project alternatives may be as good. It might be that from the developer's point of view the client-server architecture is to prefer.
As shown in Table 1 not anyone of the related architectures support all three role's requirements. Some of the user's aspects are always considered more desirable than others.

In the next section the concept of ActionBlocks and three different approaches towards the ActionBlocks are presented. As we shall see the three user roles of Jan, Carl and Minna are mirrored in these views.

<table>
<thead>
<tr>
<th>Related architectures</th>
<th>Jan</th>
<th>Carl</th>
<th>Minna</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiple trivialities</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Web presence</td>
<td>(X)</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Appliance Data Service</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Interaction spaces</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Tacoma</td>
<td>(X)</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>RFID Chef</td>
<td>(X)</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Informative things</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Invisible interfaces</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hive</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>JINI®</td>
<td>(X)</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

Table 1 Summary of interaction focus in the related architectures.
4 ACTIONBLOCKS

4.1 Approaches to ActionBlocks
What is an ActionBlock? An ActionBlock may be almost anything; a tag reader, a video camera, a projector, a button, a lamp, a loudspeaker etc. An ActionBlock may be physical interface. By manipulate physical things a user can make things happen in the virtual world and create an action in the real world.

An ActionBlock is an artifact that exists in both the physical and the digital world. ActionBlocks consist of an intelligent part and a physical part (Figure 9). The physical part contains a physical item and an action. The physical item is the part of the ActionBlock that the user can touch and see. The action is what happens when the physical item is manipulated. The computational part contains both hardware and software. It is the software that is further discussed in this report. The hardware is necessary to run the software but has less importance in this thesis. The intelligent part consists of a symbolic representation of the physical item, a logical part that make the computation and a digital representation of the action, an event. The physical item and the action have its place in the real world, but to achieve an action the computational part is needed.

Figure 9 The component parts of ActionBlocks

The emphasis may be on different parts of the ActionBlock and it is possible to discern three different kinds of focus (Figure 10).
1. Physical-Action: The physical item and the action, in other words the real world, is focused. The intelligent or computational part is just a provider.
2. Physical-Computational-Action: In this approach all components are equally focused. One think of software and physical item as one unit, an ActionBlock. It is of interest to be able to assemble the components in different combinations. It is important to be able to set up a connection between the physical part and the intelligent part in a way that supports the work. The connection means that the symbolic representation and the physical item are regarded as one, likewise with the event and the action.
3. Computational: The emphasis is on the computational part. It is the software that performs the action and the focus lies on this. From this point of view an ActionBlock may consist of only the intelligent part and the action.
4.2 Situations of Use

There’s not one approach that is the ultimate. It is natural that the approach towards ActionBlocks differs depending on the intention. There are three different types of user roles and interactions with ActionBlocks and they mirrors the three approaches above. The three roles are the user, the interaction designer and the programmer. The three roles are represented by Jan, Carl and Minna from the scenario in 1.1 Background.

The user doesn’t have to be concerned about what computational task that is performed when using the ActionBlocks. For him it is naturally to use the first approach to think about ActionBlocks (Figure 11). A physical item has a meaning. It makes something happen, an action is made. The essential with ActionBlocks is for him the physical item and the action. When he moves a physical thing it has a physical effect. For him the correspondence between the item and the action is important. Let us return to the scenario. When Jan arrives to the exhibition hall he picks up a smooth stone that he fancies from the basket at the entrance. He moves around the exhibition and at one place he hear a sound of children's laughter when they are throwing stones in the water. It remembers him of the summers in his childhood. He associates the stone to the sound file and moves along. During his visit he occasionally feels the smooth stone in his pocket and remembers the sound. Eventually he arrives to the central table and he puts the stone on the table. The loudspeakers immediately expose his favorite sound. For Jan it is the stone that is the sound. It is the action when he puts the stone at the table that make him hear the children's laughter.

The interaction designer is the person that set up the connection between the physical item, the computational part and the action. He configures the system. For him all three components have the same dignity (Figure 11). He sees the parts as components that can be combined into ActionBlocks that fulfill his requests. The interaction designer is the coordinator. He coordinates the different ActionBlocks into a system. He has a set of building blocks that he can use and combine. In the scenario it is Carl that has an idea of what the system shall be able to do. To realize his idea he collects the blocks he needs and by support of an interface combines the blocks with the software into a system that perform the task he desires. If he wants to extend the system with a possibility to associate the item to colored smoke he collects an ActionBlocks that can make smoke and add it to the system in the same way as before.

It is natural that the programmer focus on the software and what actions it can generate (Figure 11). It is he who develops the software that is going to produce the event and the action. He also handles the maintenance. To him the physical item and the action are represented in the computational part. It is the programmer...
that has the control over what actions that can be associated to a physical item because it is he that constructs the software. Let us once again return to the scenario where Minna is the programmer. Before she constructs the new pieces of software to be able to expose images, film and sounds in the exhibition hall she has got a request from Carl to do so. Minna is aware of how the software shall be used but for her it is only of importance that the system got a signal to start something and the software processes it and depending on the signal an event is produced that culminates in an action, for example a sound is played. For Minna it isn’t of importance that the central tag reader is hidden beneath the table to assign the tag reader the physical appearance of a table. It is not of interest that Jan put a stone on the table. For her it's only a tag.

All three approaches may naturally be valid for one person depending on which role he has.

4.3 Summing Up

The conclusion is that different users have different views of ActionBlocks (Figure 11). The user uses ActionBlocks in his work or like in Jan's case in his spare time. Jan makes an association between the stone and the children's laughter. He designs in use of ActionBlocks. In use only the physical part of ActionBlocks is of interest because it is the only thing the user directly interacts with.

The interaction designer designs the interaction between the system and the user and to support the user’s work. In configuration both the physical part and the computational part have to be considered to make the system achieve the intended performance. He interacts with both the computational and the physical part of ActionBlocks. Therefore the interaction designer has to be concerned of both.

The programmer designs ActionBlocks. In development only the computational part is of interest because it is the only thing the programmer interacts with.

The three ways to interact with ActionBlocks have an internal relationship. Development is needed to alter the possibilities to do configuration and use. It is a concrete platform for the other interaction forms and configuration forms a concrete platform to use, because it provides new possibilities to customize the use (Figure 12 a).

It is possible to think of the three different use situations in the other direction too (Figure 12 b). For the system to be meaningful and motivated there has to be a user that has a desire to use the system in a certain way. There must be an idea of how to use the system and that criterion is fulfilled. When the idea exists it forms an abstract platform for the interaction designer to assemble a system and the interaction designers needs form an abstract platform for what software the programmer will produce. In Figure 12 c the two situations are assembled and as you can see and probably already know is that the use is both the motivation and the

---

**Figure 11 Types of situations, roles and user approaches to ActionBlocks**

Role: **User**  
Use: **General use**

Role: **Interaction designer**  
Use: **Configuration of system**

Role: **Programmer**  
Use: **Development**  
(Maintenance and Development)

1. Physical-Action  
2. Computational  
3. Physical-Computational-Action

---

18
goal, but the interactions in the middle is equally important because otherwise the use isn't supported.

Figure 12 Internal dependences of the three use aspects

In the next chapter the three use aspects are investigated further. As we have seen in section 3 Architectures for Smart Things some of the aspects always get the higher priority than others, in the next chapter the question how to support all three aspects arises.
INTERACTION ASPECTS

The three roles introduced above interact with the system in three different ways. The interaction means designing some aspects of the ActionBlocks. The programmer designs ActionBlocks (the computational part); the interaction designer designs the interaction with ActionBlocks and the user designs in use of ActionBlocks. The interaction causes some alterations of the system, permanent or casual. There are parallels to draw between this interaction with ActionBlocks and tailoring. A tailorable system is considered designable after the system has come in use. In the article “There’s No Place Like Home: Continuing Design in Use” the authors (Henderson & Kyng, 1991) identify three ways of doing tailoring. The three possible ways are:

1. To choose between different expected behaviors.
2. To construct new behaviors out of existing components.
3. To alter the artifact.

The three alternatives has it counterparts in the three types of interaction (section 4.2). When the user designs in use he has some foreseeable opportunities to interact with the system. He can choose among them to make his task in a customized way. It corresponds to the first way of doing tailoring. For example if the user has some tags and tag readers he can combine the tags as he chooses, but he is limited to the combination that are predicted from the beginning. The user designs the system in the aspect of use. Our user Jan can associate his chosen item to objects that appeal to him anticipated that the objects are in a digital form that the exhibitor has predicted to be interesting for a visitor (Figure 13).

The interaction designer may design the interaction by combining the parts of ActionBlocks (Figure 9) and combine different ActionBlocks with each other. He chooses from existing components to do this. This interaction corresponds to the second way of doing tailoring. For example Carl chooses between existing ActionBlocks when he is to setup the system in the exhibition hall. When he misses some components they are ordered from Minna, the programmer. When he has all the components that he needs he can combine the units into a system that will satisfy the users needs (Figure 13). The interaction designer makes it possible for the user to use the system. The interactive designer designs the system in the aspect of configuration.

The programmer is the person that designs the ActionBlocks and makes it possible for interaction designer to configure a system. The programmer implements the desired behaviors by adding or altering code. Minna's task is to construct new software to support the configuration and use and if need arises she also alters the software to better fit the needs. Suppose Carl wants to make it possible to put two items at two different tag readers at the central table and in this way produce a merge of the proposed actions. This calls for Minna's skills and she has to make the logic supporting such a scenario so that such an ActionBlock may exist (Figure 13). This situation matches the third way of doing tailoring. The programmer designs the system in the aspect of development.

![Figure 13 Tailoring in context of an ActionBlocks system](image)

12 Tailoring is when a user alters an application after it has been taken in use.
The tree aspects, use, configuration and development exist in every project but as we have seen in the examples of related architectures above, the aspects are focused in very various degrees. The use aspect is naturally always important but sometimes it comes in the background to be able to focus on a special issue. Kiczalez et.al. motivated the introduction of Aspect Oriented Programming (Kiczales et.al, 1997) by pointing out that some aspects in programming is hard to grasp in procedural or object oriented programming, aspects like optimizing memory usage, to be easy to develop and maintain and synchronization of concurrent objects. One of the reasons is that these aspects cut cross the system’s basic functionality. Kiczales et.al. describes an aspect as something that cannot be cleanly encapsulated in a procedure or tends not to be functional units. The goal of Aspect Oriented programming is to support the programmer to separate components and aspects from each other, components from components and aspects from aspects to make it possible to abstract and compose them. This idea that has its origin in programming may be transferred to the use of a system. In this area it also, as we have seen, exist different aspects that are hard to combine at the same time. The interaction aspects for users, interaction designers and programmers cut across the basic mechanism in architecture, devices, software and the more soft means, apprehension of the system. By identifying the concerns it is possible to separate the aspects from each other and to focus on them one by one.

5.1 Summing Up
Up to know I have introduced several concepts of user interactions with computer systems. It might be of interest to relate these concepts to each other (Table 2). In section 4.2 Situations of Use three roles are presented; the user, the interaction designer and the programmer. These roles relate to the system in different ways (see section 4.3). The user designs in use of ActionBlocks, the interaction designer designs the interaction and the programmer designs ActionBlocks. Their interaction involves different parts of the ActionBlocks system. The user handles physical items, the interaction designer handles systems of ActionBlocks and the programmer handles the different ActionBlocks modules like the logic etc. This handling of different units has its parallels in tailoring where three different kinds of tailoring can be discerned. The first kind is when the user customizes the system from predicted behaviors to suit his needs and this corresponds to how the user handles the physical items in the ActionBlocks system. The second kind of tailoring is when the user construct new behaviors from existing components and this sort corresponds to the configuration the interaction designer performs with the ActionBlocks. The third and last kind of tailoring is when the user alters the artifact and this matches the situation were the programmer changes or constructs new software. These three aspects on interaction with the system form a foundation for a partition in three aspects: use, configuration and development.

<table>
<thead>
<tr>
<th>ActionBlocks system</th>
<th>Tailoring</th>
<th>User interaction</th>
<th>Role</th>
<th>Aspects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical items etc.</td>
<td>Choose expected behaviors</td>
<td>Design in use</td>
<td>User</td>
<td>Use</td>
</tr>
<tr>
<td>ActionBlocks</td>
<td>New behaviors from components</td>
<td>Design the interaction</td>
<td>Interaction designer</td>
<td>Configuration</td>
</tr>
<tr>
<td>applications</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ActionBlocks</td>
<td>Altering artifact</td>
<td>Design ActionBlocks</td>
<td>Programmer</td>
<td>Development</td>
</tr>
<tr>
<td>modules</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2 Summary of the correspondences between different concepts
The three aspects concern the three different roles; user, interaction designer and programmer, as shown in Table 3, represented by Jan, Carl and Minna in the scenario.

<table>
<thead>
<tr>
<th>Role</th>
<th>Views on ActionBlocks</th>
<th>Aspects</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>Physical - action</td>
<td>Use</td>
</tr>
<tr>
<td>Interaction designer</td>
<td>Physical -computational-action</td>
<td>Configuration</td>
</tr>
<tr>
<td>Programmer</td>
<td>Computational</td>
<td>Development</td>
</tr>
</tbody>
</table>

Table 3 Correspondences between roles, relations to ActionBlocks and aspects

The aspect *use* concerns Jan and he is focusing on the physical part and the action of ActionBlocks. For Jan the real world is the essential. Carl that is interaction designer is concerned about the aspect *configuration* and he thinks of the computational part and the physical item as an undivided ActionBlock. The third role, Minna, is involved in the *development*. Her view of ActionBlocks is that it’s the computational part that is most important. An ActionBlock may even be only software (Figure 14). This indicates that the three aspects will be affected by these views. The views are to be reflected in the software architecture for ActionBlocks.

Figure 14 The three aspects in context of ActionBlocks
6 ARCHITECTURES

We have three different ways of relating to the system; use, configuration and development. These three aspects might need different architectures. To cover a spectra of architectures I have chosen to exemplify the correspondence of the three aspects and the architectures by pure Peer-to-Peer architecture, Peer-to-Peer with distributed services and client-server architecture though it covers the extremes and an example in the middle.

6.1 Pure Peer-to-Peer

In the first approach, use, it is important that the ActionBlocks as physical objects are in the front. The user shall not be bothered with concern over the computational part. It should be embedded in the ActionBlocks, to make the user able to associate the physical item directly to the action. In a pure Peer-to-Peer architecture (Figure 15) the nodes reside all the computation, location of other nodes etc. In pure Peer-to-Peer architectures all the peers have the same dignity. Every peer handles its own affairs and when it needs support from some other peer it makes contact and the contacted peer may reject or accept the peer's request. In other words the action the ActionBlocks perform is set in the ActionBlock itself. The advantages with pure Peer-to-Peer are scalability and the system gets robust because if a peer breaks the network may continue to exist. It is also easy to join the network. It is just to start a peer (Minar et.al, 1999). These advantages are typical user requirements. It makes the system easy to use. The correspondence to the first aspect is total. There are also some disadvantages with Peer-to-Peer. It is hard to keep a list over existing peers in the network because all peers hold their own lists and they may not be complete. It may also be a problem to administrate different versions of software components (Minar et.al, 1999). The disadvantages concern the configuration and development aspect.

![Figure 15 Pure Peer-to-Peer architecture](image)

6.2 Peer-to-Peer with Distributed Services

In the configuration aspect all parts in ActionBlocks is equally focused. The main thing is that the designer may configure his ActionBlocks to suit his needs. The appearance of the physical item is of importance as well as the performance of the action. The interaction designer must be aware of the computation to be able to configure the ActionBlocks. It is less important were the computation take place, it is much more of interest that the computation is at place at the right time. The ActionBlocks may receive help from nodes with extended functionality. Such architecture is Peer-to-Peer architecture with distributed services (Figure 16). The distributed services may load the needed logic to the required ActionBlock or locate other ActionBlocks. The approach opens up for customized solutions were
some services can be centralized but the communication between the ActionBlocks is direct. The architecture is an effort to minimize the disadvantages with Peer-to-Peer and client-server architectures.

![Figure 16 Peer-to-Peer architecture with distributed services](image)

6.3 Centralized Architecture

In the development aspect, the computational part is in focus. The physical item is of minor importance. It exists as a symbolic representation within the software. With this approach it is convenient to make all severe computation at one place, at the server. The server controls the network and holds the resources that is shared between the clients. Usually, the processing is done by the server but the processing may also be shared between the server and the client. For example a server can search a database upon a request of the client and the result is sent to the client. Then the client makes some calculations upon the result from the database search. From the programmers point of view the clients are input devices that make requests to the server that serves the client with the required service. For example, coordinate the communication between different ActionBlocks. An advantage with a client-server architecture is that it reduces the data traffic on the network (Capron, 1997).

It is much more easy to maintain and update a system with client-server architecture (Huang et.al, 2001) where many of the resources are kept at one place. It is also easy to keep a list over existing clients (Minar et.al, 1999). The development aspect indicates that a centralized architecture, client-server, may be to prefer as architecture for ActionBlocks (Figure 17). The disadvantages with client-server architecture concern the use aspect. When the server breaks, the network ceases to function and there is no personal control over the server. Another disadvantage is that the user's apprehension of ActionBlocks doesn't correspond to a client-server architecture where the ActionBlocks just are dumb devices that rely on the server.

![Figure 17 Server – client architecture](image)
6.4 Summing Up

The three architectures have as we have seen its counterparts in the three aspects use, configuration and development.

In the use aspect it is important that the system is easy to use and is robust. It is also of importance that it is possible to move the ActionBlocks at the Internet. It is important that the system is scalable. But it is also of great significance how the user apprehends the system. In use the center of attention is on the physical part of the system. The user likes to think about the ActionBlocks as physical objects and the desired functionality is preferable embedded into the physical object. The interaction shall have a physical appearance. The communication between the ActionBlocks will be direct and if the connection breaks between some ActionBlocks the network will still function. If a break appears it is one of the ActionBlocks that fails, not a server somewhere at the Internet. Therefore the software architecture for ActionBlocks will be a Peer-to-Peer architecture when the system is in use.

In the configuration aspect the architecture is of less importance. The main thing is that the configuration is uncomplicated. When the interaction designer configures the system he isn’t especially concerned about the architecture. It might be a client-server or a Peer-to-Peer architecture. A Peer-to-Peer architecture with distributed service has some advantages from both those architectures and that's why this architecture is chosen for the configuration aspect.

In the development aspect it's convenient to have most of the software collected at one or at any rate only a few places. It is much more demanding to maintain similar software located on many different peers. This speaks for a client-server approach from the programmers view.

We now have three different suggestions for architectures for ActionBlocks. One for each aspect. In the next section I will focus on one aspect at a time.
7 CONCEPTS FOR ARCHITECTURES

7.1 Requirements of the Architecture

There are some requirements to fulfill in future software architecture for ActionBlocks. The requirements have a starting point in wishes and opinions of the personnel at Interactive Institute and have been guidelines for the design of the concepts. The requirements can be summarized in following points:

**Use**
- The system shall be easy to use.
- The system shall have good operability.
- At unpredictable breaks the users will only be involved at a limited range and only easy manipulation may be needed.
- ActionBlocks may move freely at the Internet.
- It shall be possible to, for a moment, turn of an ActionBlock in a configuration.
- The system shall be scalable.
- Even if ActionBlocks may be located behind a firewall a connection shall take place.

**Configuration**
- Different configurations and several users may use the system simultaneously.
- ActionBlocks shall be a part of Internet.
- Different input may be combined to create an action together.
- It shall be possible for the interaction designer to combine different ActionBlocks and logics in an easy way.

**Development**
- New types of ActionBlocks may be added without altering any code.
- The system shall be easy to maintain and develop.
- ActionBlocks may be both in and out device.

7.2 Use

In use there are some ActionBlocks that the user interacts with and some ActionBlocks that produce actions. The user interacts with the ActionBlocks, for example puts some tags on the tag reader or pushes a button. The receiving ActionBlock creates an event and an action is made, for example a camera is activated.

Figure 18 Interaction between ActionBlocks in the aspect of use.
The ActionBlocks may be combined in various combinations and it is the logic that makes the combination possible. It is understood that the ActionBlocks are a part of the Internet. In Figure 18 the user interacts with two ActionBlocks and another ActionBlock produces the action. Let us say that the two ActionBlocks to the left are two tag readers and the ActionBlock to the right is a monitor. When the user put tag 1 and tag 2 on the tag readers a Donald Duck film is played and if tag 2 and 3 are at the tag readers "Titanic" is played. The logic for what film is shown lies at the monitor.

Let us return to the scenario in 1.1 Background. When Jan has associated his stone to the sound he arrives to the central table. A tag reader is hidden beneath it. Somewhere in the exhibition hall a projector resides. It will project a film or an image on the big screen. Other things that are essential are the loudspeaker and the spotlight switch. The constellation of ActionBlocks may look like in Figure 19.

This scenario differs from the example in Figure 18. In this scenario there is just one ActionBlock that Jan interacts with and three ActionBlocks that produce an action. From Jan's point of view it is the tagged stone that make something happen. It is his action to put the stone at the table that makes the physical loudspeaker to act. To Jan it is the table (the tag reader) that knows what to do when the stone is put at the table (the specific tag is put on the tag reader). What happens is that the tag reader sends the tag id to the loudspeaker, which decides depending on the tag id, which sounds to play. In Figure 20 the way the messages takes is shown. The red doted arrow and the red circles represent what the user apprehend.

Figure 19 ActionBlocks in the exhibition hall

Figure 20 Schematic picture of how the message travel
The gist of these examples is that the ActionBlocks know by themselves what to do. They don’t have to take help from some central server that holds all the information.

7.2.1 Terminology

The tag readers, loudspeakers and monitors in the examples above are just instances of the more general term ActionBlock. Every type of ActionBlock has a specific functionality. A tag reader sends tag id, a loudspeaker exposes sound, the spotlight switch lights different spotlights and a monitor displays graphical objects for example. To be able to speak of ActionBlocks in a more general way I introduce transmitters and receivers. A transmitter is an ActionBlock that has as a task to send tag id or other types of translated signals to a receiver that performs an action. An ActionBlock may act as both a transmitter and a receiver at the same time if such a circumstance occurs.

In the example with the two tag readers and a monitor in Figure 18. The monitor might not be able to store the films because of lack of memory space. It has to make a request to some type of database. Several ActionBlocks might share this database. This database isn't really regarded as an ActionBlock due to the definition of ActionBlocks (4.1 Approaches to ActionBlocks) as a unit that contains an intelligent and a physical part. At least it doesn’t have a physical part that the end-user comes in contact with. But this type of unit has to exist and to be able to speak of this type in a general way it goes under the name of responder. A responder doesn't have to be a database it can also be a unit that makes a computation that is shared among several ActionBlocks.

7.2.2 Concept

The conclusion of the discussion in section 7.2 is that in the use aspect all ActionBlocks has to have the software needed to perform its task corresponding to its functionality so that it can act by itself. Transmitters need software that make them translate an electric signal from for example a tag and send a message further with the tag id to a specific receiver. A receiver shall be able to compute on the received data to decide what action to make and then make the action. The architecture is to be Peer-to-Peer.

It shall also be possible to turn of a certain receiver. Suppose there is going to be a short lecture at the exhibition hall in the scenario. Then it isn't desirable that there are visitors that put items on the central table to make sounds or films. And it is neither wanted to have to turn the whole system down. It shall be enough to put a specific tag on the tag reader to turn the receivers functionality of, or in other words make the receiver pause its actions. In this way it is possible to only turn one of the receivers of, if that is required.

7.3 Configuration

To be able to use the system, the use has to be proceeded by a configuration. The participant in this aspect is the interaction designer. Once again we return to the scenario. Carl is to configure the system. He has decided to configure the system before he plug in the ActionBlocks to the network. He could configure them at place, but he doesn’t have access to the exhibition hall at this time therefore it is easier to have the ActionBlocks disconnected. At a shelf at the Space studio he collects the blocks he needs and starts the configuration application at a computer. Next to the computer a tag reader is connected to the network. When Carl starts the program the interface shows a working space where all the available logic units (software) and responders already are shown. He puts his chosen ActionBlock one by one on the connected tag reader and they are immediately shown in the interface as icons. This can be done by the fact that all the ActionBlocks are tagged. When he is finished he can associate the different ActionBlocks and logic units to each other (Figure 21). The responders and the logic is dragged and dropped in the
middle of the interface and then Carl can bind them together the way they are going to communicate.

Figure 21 Schematic interface to configure a system (Original idea Peter Warren)

7.3.1 Terminology

The configuration has a special position in the concept suite. The interaction aspects affect the system when it’s in two different states, when the system is active (operating) and passive (not operating). The use aspect involves the system when its active and the development aspect involve the system when it’s passive. The configuration aspect scopes both these states. I separate the configuration into two parts: the system configuration and the setup (Figure 22). The system configuration is exemplified by the configuration Carl does above. The setup is invisible for all roles. It is performed automatically by the system when the application is started. When the setup is performed, the ActionBlocks are connected to each other and prepared for use. At the setup a computer with extended services is needed. On this computer the configuration service is running. This service helps the ActionBlocks in the application to be aware of which other ActionBlocks to communicate with. When the different ActionBlocks has this information they can be used. At setup the system is active and when the ActionBlocks are associated, configured, the system is regarded as passive.

Figure 22 System states in context of interaction aspects
Some would certainly argue that setup is a part of use, but I prefer to consider it as configuration while it is a preparation for use. Starting the car is just a preparation for driving. Starting the car is not really considered as driving.

To make the configuration the interaction designer has to use some kind of interface. The configuration program might be installed on several computers or perhaps it is web based. That is of minor importance to the interaction designer. A web based application has the advantage that he can configure a system where ever he is, provided that he either has a tag reader connected to the network available or the wanted ActionBlocks already are plugged into the network. Such a program that supports configuration or other types of extended use I call *appliances* to be able to talk about them generally. Such an appliance is used in the scenario when Jan associate his stone to the sound of laughing children, this specific appliance is called the *tag configuration appliance* and it supplies an interface for Jan to use. The *tag configuration appliance* uses the *tag configuration manager* to retrieve the data that is needed. When Carl configures the system he uses another appliance called just *configuration appliance*. The *configuration appliance* uses the *configuration manager* to get knowledge of the representations of the tags etc. It is the same relationship between system in use at system configuration and the system to be constructed, as the egg and the hen or compiling code with a compiler written in the same language.

When the interaction designer has associated some ActionBlocks to each other, configured a system, he has composed an *application*. The term application is both physical and digital in this meaning unlike in ordinary use of the term.

### 7.3.2 Concept

#### System Configuration

It is practical to think that the interaction designer configures the system by a drag-and-drop interface (Figure 21) at the computer screen just like in Hive (Minar et al., 1999).

All ActionBlocks are tagged and can therefore be shown as icons in an interface accompanying the *configuration manager* (section 7.3.1). To make it possible to show the ActionBlocks in the interface, tag id together with a description or image of the ActionBlocks has to be obtained in a database. If the tag is known, the icon is shown. The available logics and responders are shown from the beginning, as mentioned earlier (section 7.3). These units are not shown in Figure 23.

When Carl put his chosen ActionBlocks at the tag reader (1)¹³ (section 7.3) the following things happen in the *configuration manager*: The tag reader sends a message to the manager with the tag's id (2). The manager sends a request to the Service provider that looks the tag id up to get knowledge of what ActionBlock that this tag is attached to (3). After that is done the description of the ActionBlock is picked up (4) and the symbol may be displayed at the interface (5). Carl continues the process to put different ActionBlocks at the tag reader (1-5). When he is ready with this he starts configuring the system. Eventually the description of the application Carl has produced is stored in a database that can be used later at setup (6-7). But before this is done the reconnection principles has to be set. The reconnection principles are used at the setup to determine if the right ActionBlocks are available.

¹³ The number refers to the numbers in Figure 23.
The interaction designer has to choose what reconnection principles that he wants for the configuration, if he wants the ActionBlock ids to matter, if the location of the ActionBlocks is important etc.

There are four reconnection principles: Logic scheme, Location scheme, ID scheme and ID and Location scheme (Figure 24).

Let us take an example. We have done a configuration earlier and it contains three ActionBlocks. Two tag readers (id: A and B) and a camera (id: 1). One of the tag readers is at location 1 and the other at location 2. The camera is placed at
location 3. The interaction designer has chosen the reconnection principle to be 4, 3, 2, 1 (the numbers refer to the numbers in Figure 24). When the application is to become reconnected the first control is to check if the available ActionBlocks has the same id and location as the original configuration. If this isn’t possible the possibility of the third scheme is checked to see if there are ActionBlocks that have the right id. If that is the case those ActionBlocks are selected. But if there isn’t a match the second reconnection principle is checked. Is there ActionBlock of the right type in the right locations? If so those ActionBlocks are chosen. If not, the logical scheme is the only one left to check. The only thing that is important is that the ActionBlocks are of the right type. When the configuration is done and the reconnection principles are set the values are saved.

The architecture may be regarded as a Peer-to-Peer architecture with distributed service because the tag reader is a peer and the computer with the configuration appliance is a peer and the configuration manager is the distributed service that may be on the same computer or at another one.

Setup

The setup’s first step takes place when the ActionBlocks are plugged into the network (Figure 25). The ActionBlock starts, get registered as active at the configuration service and get prepared to communicate with other ActionBlocks when an application is started. This can be done by putting a dedicated tag on the tag reader or if the application has a GUI, by starting it from there. One can also think that an ordinary physical button can start an application. The options are many.

Let us imagine that Carl starts the exhibition application by putting a tag on the table in the exhibition hall. What will then happen behind the curtains (Figure 25)?

The first thing that will happen is that the tag reader will send a start message (1)\textsuperscript{14} to the configuration service. The service looks up the application to get knowledge about which ActionBlocks and which logic that are required (2), and then it is checked that the required ActionBlocks are available (3). It might be the case that several applications (4) are operating at the same time if so the communication between the ActionBlocks has to be coordinated. If everything is all right so far the

\textsuperscript{14} The numbers are referreeing to the numbers in Figure 25.
needed logic is located (5). The last step is to inform all the ActionBlocks about which to communicate with and to upload needed logic (7.4 Development) to the appropriate ActionBlocks (6).

**Tag Configuration**

There is another phase of the scenario in section 1.1 that isn't processed yet. It is when Jan picks up an unidentified tag and associates it to his favorite sound. Jan performs this association in an appliance called the *tag configuration manager* (Figure 26). The GUI contains a representation of a tag reader. When a tag is put on the tag reader it is shown in the interface. When Jan puts his tagged stone at the tag reader (1)\(^{15}\) a message is sent to the manager (2) that get knowledge of the tag id. The tag reader representation indicates that a tag is put on the physical tag reader (3). The available files are shown in the explorer and when Jan knows the name of the file he wants, he can click on it and drag it to the tag reader in the interface (4). The manager sends a request to the service provider that fetches a representation from the file description storage (5) and displays it on the digital tag reader. It might be a picture with laughing children in Jan's case. Jan saves the setting (6) and the tag id and the file is then associated to each other and stored in a database (7). As you can see this *tag configuration manager* and appliance is quite similar to the *configuration manager* and appliance in Figure 23. The interface in this appliance is really a sub-interface to the configuration interface. The appliance is a *configuration appliance* where the interaction designer configures tags to represent different ActionBlocks to be able to configure the ActionBlocks into applications. But it can be used as a user appliance too and this is what is done in the scenario. When it is used as a user appliance and associated to files instead of ActionBlocks the *Tag → File* storage is used instead of the *Tag → Type* storage. It makes sense to separate them when the *tag configuration manager* is used as a user appliance because among other things the user's associations is only temporary and it isn't always useful to keep them for a longer time.

---

\(^{15}\) The numbers are referencing to the numbers in Figure 26.
7.4 Development

The development aspect is concerned about maintenance and development of the system. Let us take a look at Minna’s task in the scenario. When Carl ask Minna to make some new logic for the exhibition she has to construct three pieces of software; one for the spotlight switch, one for the loudspeaker set and one for the projector. Then the projector has never been used in an ActionBlocks system before and therefore there is a need for some general software in it.

What Minna has to do is to equip the projector with an event module and a symbolic representation. The architecture for the ActionBlocks define that the part called Event (Figure 9) handles the event generated by the user and the events from the network. That part is the same for all ActionBlocks, so that piece is just to upload to the projector. The symbolic representation has to be constructed. This representation is divided into two parts. It contains the software that handles the basic functionality for an ActionBlock, for examples signals from tags has to be translated to a form that the logic can understand. This part is specific for each kind of ActionBlock (tag reader, button etc.). The other part is a description of the ActionBlock, what type it is (receiver, transmitter), what sort it is (tag reader, projector), name, id etc.

When this is done Minna turns to constructing the logic for the projector, spotlight switch and the loudspeaker set. Minna has to make her new software adapt to a predefined interface to the event module that handles the communication. To save time in the future she makes the film and image displays etc. general so that they can be used in another situation later. This is often done by programmers, but there is one more thing to think about. The use of ActionBlocks differs during time and no project is similar to another therefore it has to be easy to put different logics together. Therefore the logic units also have to be adapted to a predefined interface towards each other. When Minna has finished her work with the logic she saves it in the logic storage (Figure 27) in the configuration service. When Carl configures the system the logic will be associated to the application and at setup uploaded to the intended ActionBlock.

So then, Minna’s task is to code new logic units and to load them to an ActionBlock or the computer that have the configuration service (Figure 25). When Minna is upgrading the system is passive, not operating.

7.4.1 Concept

Let us take a closer look at the projector logic. What is needed? The projector is going to show images and films. In another situation only displaying images might be needed. It seems to be convenient to make these two abilities in separate software that could be combined. But to be able to make the decision what type of media to display a third logic unit is needed, a part that holds the information about which tag is associated to which file (Figure 28), the direction guide (this part might get assistance from a responder, but we disregard that possibility for this time). The same principles go for the loudspeaker and the spotlight switch.
If only the logic for displaying images will be used in a monitor the logic shall be able to be reused (Figure 29). The direction guide will be different. To be able to combine logic in this way the various logics have to adapt to a general interface.

To make the programmer’s task easier the different logics are uploaded to the intended ActionBlocks. The only thing the programmer has to do is to put the logics in the logic storage. In this way the programmer is able to look at the computer that stores the logic as a server. When the interaction designer configures the system he chooses what logic to use and when he saves the application it registers what logic and ActionBlocks the application require. In this way it is possible to upload the right logic to the right ActionBlock at setup.

The second part of the programmer’s task is to supply the ActionBlocks with the Event and the Symbolic representation. To the programmer these ActionBlocks are regarded as clients. The event is the same for every ActionBlock. It takes care of the communication between the action and the logic, but it is the Symbolic representation that besides wrapping the specific data corresponding to the specific ActionBlock defines how the physical action is to be translated. The symbolic representation differs from ActionBlocks type to ActionBlocks type. The data part of the symbolic representation differs for each ActionBlock (name and id for example).

The programmer may regard the system as client-server architecture. It may not be what one typically regard as a client-server architecture, while the ActionBlocks is independent when the system is in use. The ActionBlocks just require assistance from the server (configuration service) during the setup, when a request for starting
an application is made. The configuration service responses by sending messages to the ActionBlock about which to communicate with.

A client is a equipment or software that require services from a server and a server is a function in a network that serves other nodes (Berti, 1998). The server also holds shared resources and processes the requests from the clients (Capron, 1997). From the programmers point of view an ActionBlock is a piece of software that needs assistance from the configuration service to be able to start an application. An ActionBlock also need logic that is stored in the Logic storage at the configuration service. The configuration service holds for example the data of which ActionBlocks and logic that is required for an application and it processes the available ActionBlocks and the required ActionBlocks to find out if an application may be started. This together with the fact that the system is not operating when the programmer interacts with the system, makes it possible for the programmer to regard the architecture as a client-server architecture in spite of the fact that the system doesn't act as an client-server architecture in use.
7.5 Summing Up

Now we got three concepts one for each use aspect. The challenge now is to combine these concepts into one, so that every role gets the most appropriate architecture when interacting with the system.

By combining Figure 19, Figure 23, Figure 25 and Figure 30 the three aspects of the system get synthesized into one complete system. Figure 31 shows the whole system. We can see that it ended up to be a loose coupling between the three parts in the suggested architecture.

Figure 31 The whole system
8 PROTOTYPE

The prototype presented here is a first step towards an application founded on the proposed architecture that has seven ActionBlocks and one appliance. Six ActionBlocks are tag readers and the other one is a loudspeaker. The application is a part of a bigger project (ATELIER) and is a demo to show some possibilities with the technology in the scope of design environments. The project will continue for two years. The demo is a kind of sound player where the user plays different sounds by manipulating physical objects. The user can associate different sound files to tagged objects in a *tag configuration appliance* and then play with the object by putting them on the keys (TR₁- TR₄ Figure 32). By putting a tag on one of the control tag readers (TR₅- TR₆ Figure 32) the sound may differ in speed or be played in a sequence or all the sounds at the same time.

An ActionBlock contains five different parts (Figure 9): a physical item, an action, an event, logic and a symbolic representation. The physical item and the action are not further discussed because they reside in the real world not the digital. The three others may be divided in subparts (7.4.1 Concept). The ActionBlocks architecture is meant to suit different types of applications and it is therefore natural to divide the work with the application in two parts one that handles the communications and the events and one that handles the logic and appliances. The work with the application is a collaboration between my external advisor Peter Warren and myself. Peter and I have divided the work so that Peter makes the logic and the appliance while I concentrate on the event module. That leaves the symbolic representation. For tag reader there already exists a piece of software that translates the signals from tags so that part is disregarded for now. The same thing goes for the loudspeaker. Peter already has software for the functionality for the loudspeaker. The description of the ActionBlock that also is a part of the symbolic representation is used by both of us. Therefore we have developed it together.

The event module is general for all types of applications while the logic and the appliances may differ. The description of the prototype will therefore concentrate on the event module and the interface between the event module and the logic.

8.1 Platform

Many of the requirements (7.1 Requirements of the Architecture) can be associated to the communication between ActionBlocks. Requirements like:

- ActionBlocks shall be a part of Internet.
- ActionBlocks may move freely at the Internet.
- The system shall be scalable.
- The system shall have good operability.
• Even if ActionBlocks may be located behind a firewall a connection shall take place.

For example to make ActionBlocks find each other at Internet without knowing in advance where to find each other isn't an easy achievement. A solution might be that they always communicate over a server that know where to find them, but because the use aspect require an Peer-to-Peer architecture a decision to use JXTA (www.jxta.org) as a platform was made. JXTA is a Peer-to-Peer platform that has the ability to find other peers at Internet and to make a connection to a specific peer. JXTA is an open source project that started at Sun Microsystems, Inc. and there is an implementation in Java to use. This suits my purpose well.

8.1.1 JXTA

The shortening JXTA come from juxtapose that means side by side. JXTA is a set of Peer-to-Peer protocols that make it possible for different kinds of devices on the network to communicate with each other and to work together as peers. The protocols create a virtual network between the participating peers (Traversat et.al; 2002). The devices can be cell phones, PDAs, PC or server etc. JXTA is language (C, Java etc.), system (Windows, UNIX etc.) and network (TCP/IP, Bluetooth etc.) independent and it exists implementations in several languages (Gong, 2001). JXTA is also designed to be available for small devices used in ubiquitous computing. The JXTA project started as a research project at Sun Microsystems but is now open source for all to join and contribute to (www.jxta.org).

The protocols make it possible to discover other peers, make peers organize in peer groups, discover and publish different network services, communicate with other peers and to monitor other peers. JXTA also provide a possibility to discover other peers across firewalls. To achieve this one peer inside and one peer outside the firewall has to be aware of each other. Such a peer that forward messages are called relay (Figure 33). The two peers use HTTP transfers (Java Programmer's Guide).

Another type of peer is a rendezvous peer that forwards requests and help other peers to discover requested resources (Sun Microsystems, Inc).

There are several key issues in JXTA and among these peer, peer groups, advertisements, pipes and messages are especially essential in the prototype.
• Peers organize themselves into peer groups, a virtual community of selected peers.
• A peer advertises its services through an advertisement, an XML document. The advertisement makes it possible for other peers to discover which services the specific peer can contribute with.
• Peer uses pipes to send messages to each other. A pipe is bound to a specific endpoint for example a TCP port. There is input and output pipes and they are asynchronous.
• A message is a XML document. The envelope contains among other things routing and credential information (Gong, 2001).

When two peers want to communicate they joins the same peer group. Let us take a simplified example: If peer A wants to communicate with peer B, peer A locate the peer group advertisement for the peer group that peer B is a member in. When the peer group advertisement is cached peer A applies for a membership and joins the group. Then the pipe advertisement for the group has to be located and cached.
When this is done the peers creates one pipe each based on the pipe advertisement. The created pipe might be an input or an output pipe depending on if the peer is going to send or retrieve information. Of course there is more to it than this but let us keep it simple.

8.2 Limitations in the Prototype
Due to the limited time for developing the prototype it has to be limited to scope the most desirable requirements. Some of the requirements are taken care of through JXTA, such as that the ActionBlocks shall be a part of Internet. But others have to wait until further work. In the use aspect all the requirements might be satisfied. But in the configuration aspect some limitations has to be made. The configuration appliance mentioned in 7.3 Configuration is a fulfillment of the point below.

- It shall be possible for the interaction designer to combine different ActionBlocks and logics in an easy way.

The appliance and the rest of the program associated with the appliance will not be implemented. Responder are not supported either. Another thing that relate to the configuration aspect is the ability for the ActionBlocks to find each other if the configuration service is not operating. It is my intention that the ActionBlocks may contact other ActionBlocks that were members in the previous application by themselves, but this isn't done yet.

An unfortunate thing is that there has been a great shortage of tag readers. Many different unlucky circumstances have done that I have been forced to simulate the physical tag readers with software representations. But this fact doesn't affect the validity of the principles.

The prototype presented here implement the event in following aspects:

- The use aspect.
- The setup in the configuration aspect.
- The development aspect.

8.3 Use
When a peer sends a message to a group all group members get the message. To avoid that all messages are sent to all peers in an application (tag readers isn't supposed to talk to each other) several peer groups are used in each application. They are called communication groups in the prototype. Each communication group can only contain one transmitter but several receivers (Figure 34). An ActionBlock either sends or listens at a communication group that is related to one pipe.

![Figure 34 One application and three communication groups.](image)

When the application is ready to be used the communication groups and the pipes already are created. The receivers' listen on pipes and the transmitters send on one pipe each.

When the user puts a tag on the tag reader (1) \(^{16}\) reads the tag id and perhaps other data that the tag contains (Figure 35). The signal from the tag reader is then

\(^{16}\) The numbers refer to the numbers in Figure 35
translated so that the logic can understand it (2). After that the data is translated into a XML document (Appendix I) in the logic module and the document is submitted to the interface (called IPP\(^{17}\)) (3) between the event and logic module (In the prototype this part is simulated by a software tag reader). The IPP sends the XML document (as an InputStream) to the transmitter (4); the transmitter forwards it to the communication group (5). The XML document is wrapped up in a JXTA message and sent on the output pipe (6) related to the communication group. One or more receivers are listening on the pipe (7) and when the message with the tag data arrives the XML document is unwrapped and forwarded to the receiver (8-9), which sends an update to the IPP (10) that reacts and inform the logic (11). The logic processes the data as it pleases and produces an action of some kind that corresponds to the ActionBlocks basic functionality (plays a sound if it is a loudspeaker) (12-13). To send a message explicit method calls are used, because the objects have knowledge of the other in turn. But when a message arrives such a relationship doesn’t exist naturally. Instead the classes observe the others and when a message arrives the object in turn get knowledge of it. The received message travels backwards through the object chain.

Figure 35 Schematic dataflow when the user put a tag on the tag reader.

The interface called IPP (Inter Process Protocol) has one entrance and one exit. The only thing that travels across the IPP is XML documents in an InputStream. In the event module the document is wrapped in a JXTA message and in the logic it may be wrapped into any object suited for the purpose. The logic and the event module treat the XML document in its own way without any interference of the other part. They are completely independent. The construction of a transmitter and a receiver is the same. It is the description of the ActionBlock that restrain the functionality for the two types.

8.4 Configuration

8.4.1 Setup

When the ActionBlocks in the prototype start they all do the same thing. They start the JXTA platform, check which IP-address they have (The ActionBlock may have been moved and the IP address has to be known to get out into the network.), locate the "IPP_HomeGroup", which is a peer group where all active ActionBlocks

\(^{17}\) Inter Process Protocol
are members, and joins the "IPP_HomeGroup". Then the "IPP_HomePipe" is discovered and an input pipe created. On this pipe the messages about which group to join come.

The symbolic representation consists of a description of the ActionBlock (Appendix II) as mentioned earlier (section 7.4.1) and it is from this document the ActionBlock get knowledge of which home group to join.

After these tasks are achieved the ActionBlocks are ready to be a part of one or more operating applications but first somebody has to make a request to the configuration service to setup communication channels for a specific application. When such a request arrives the configuration service starts to work. At this time the configuration service already has made the same preparations as the ActionBlocks. It has joined the "IPP_HomeGroup" and because it is a configuration service it has created an output pipe related to the home group. It has also discovered which ActionBlocks that are active and saved the information in a XML document called ActiveActionBlocks.xml (Appendix IV). The configuration service checks periodically the active ActionBlocks. When the request to start an application arrives, the SoundPlayer for example, the configuration service checks the XML document called SoundPlayer (Appendix III), that contains data about the application, to get knowledge of which ActionBlocks that are required and what reconnection principle (section 7.3.2) that has highest priority. The data was assembled and saved from the configuration appliance the interaction designer uses to configure the system (section 7.3). Then the configuration service checks if the required ActionBlocks is available. If the required ActionBlocks are available there is a control to see if they are allocated by another application. If not so, the procedure continues otherwise the user gets informed that there are ActionBlocks missing. The active applications are stored in an XML document called ActiveApplications.xml (Appendix IV). The configuration service creates a new group and pipe advertisement for each transmitter in the requested application. Then messages with the group data (Appendix V) are sent on the "IPP_HomePipe". Each message has an addressee so that only the intended ActionBlock accept the message. Next step is to upload the needed logic to the ActionBlocks in question. What logic goes where is described in the application description (Appendix III). Eventually the application that is just started is saved in the ActiveApplications document to be used later if another application is to be started.

When the messages about which groups to join arrive to the different ActionBlocks, a transmitter joins one group and a receiver joins all the communication groups. And dependent on if the ActionBlock is a receiver or a transmitter an input respectively output pipe is created for the group. It may be a message with logic that arrives instead. If so the ActionBlock receives the logic and saves it in a dedicated catalog. The application is now ready to be used.

**Recapitulation of the setup at the configuration service:**

0. Starts JXTA, joins home group and creates output pipe.
1. Discovers active ActionBlocks and saves them in ActiveActionBlocks.xml
2. Checks the XML document that contains the application description to see if the required ActionBlocks are available.
3. Checks which applications that are active in ActiveApplications.xml.
4. Checks if some of the required ActionBlocks are active in another application, if so checks if they are exclusively allocated.
5. For each transmitter a group and pipe advertisements are created and a message is sent to each transmitter to join the specific communication group. Messages to join all the communication groups are sent to the receivers in the application.
6. Uploads the logic to the intended ActionBlock.
7. The started application is saved in `ActiveApplication.xml`.

**Recapitulation of the setup at the ActionBlock:**

0. Starts JXTA, joins home group and creates input pipe.
1. a.) When a message from the *configuration service* arrives, the ActionBlock reads the message, if it is addressed to it, and joins the imposed communication group.
   b.) The alternative is that it is a message with logic that arrives. Then the logic is dealt with and saved to the disk.
2. Depending on which type the ActionBlock is an output or an input pipe is created. If the ActionBlock is a receiver an input pipe is created and if the ActionBlock is a transmitter an output pipe is created. The classes are the same. It is the XML document that controls this process.

**8.5 Development**

When the development aspect is actualized the system is not operating, but the result of the development (the logic) is uploaded to the ActionBlocks while the system is active. It is uploaded at setup. How the upload is carried out is described in this section.

When the programmer construct the logic it has to adapt to the IPP interface between the event and the logic as well as to the interface between the basic functionality in the symbolic representation. Another thing that is essential when constructing new logic is that the XML documents that travels through the interface is well defined so that different logics are able to understand them. An advantage with XML documents is that unimportant data can be ignored. But it is important that all logics can understand the parts that they suppose to. Therefore the document can't vary from time to time.

There is at least two ways to upload and integrate new logic to an ActionBlock. The most obvious and convenient is for the programmer to construct the logic, name it, compile it and store it. When the logic then is to be uploaded the *configuration service* read the files into InputStreams and sends them with a JXTA message to the ActionBlock. When the logic is to be used the classes are dynamically loaded into the program. This is the most natural way to proceed while it exist standard methods in Java API to do so.

Although, there is an alternative to dynamic loading of classes. Both the JXTA and the implementation of the architecture are founded on different descriptive XML documents. This idea can be used for the logic too.

To be able to avoid dynamic loading of classes the program has to know about the logic in advance. The well defined interface towards the event (IPP) and the basic functionality in the symbolic representation makes this possible, as long as the logic always has the same name and adapt to the interfaces. Naturally it isn't possible to store several files with the same name at the same place. That means that the files have to be renamed before saving and then just before uploading. This procedure seems just a little bit too artificial and longwinded. Instead when the programmer is finished constructing the logic he puts the source code into a XML document. That takes care of the naming problem while the java file in the XML document has the same name as all logics but the XML document can have a name that describes the abilities of the logic. When the logic is to be uploaded the whole XML document gets wrapped up in the JXTA message. When the message arrives to the ActionBlock the XML document is parsed and the string with the source code is saved in a file with an appropriate name. The source file is then compiled at place, during runtime. Such a method exists in `java.lang.Runtime`. The logic is then ready to be used.
8.6 **Summing Up**

8.6.1 **Use**

The concept of the architecture has to fulfill some requirements (7.1 Requirements of the Architecture). During the use aspect the system shall be easy to use for example. This is achieved by the fact that everything that can be automated in a natural way is automated. For example shall the ActionBlock start up automatically while plugged into the network. In the setting described above the user just has to decide which application to start and then the rest will be handled by the configuration service and the ActionBlocks themselves. How the start command is given is up to the application. It might be started by a tag or by pushing a button, physical or in a GUI. How easy and natural the system will be to use is also up to how well the association with the physical objects (tagged) and the digital representation match.

If the application cease to function during use the fault lies in the communication between the ActionBlocks or in the ActionBlocks itself. If such a severe malfunction arises so that one of the ActionBlocks breaks, then there isn't much for the user to do. But if it is the communication that is broken the user can pull out the cable and plug it in again. Then the configuration service makes new connections or the ActionBlock itself contact previous members in the group to continue the communication.

JXTA's ability to build virtual networks makes it easy to discover other peers on the network wherever they are, provided that they are JXTA peers. This makes it easy to move the ActionBlocks around. The only thing the ActionBlock has to do is to check its own IP-address and to see if it has altered since last time. But dependent of the network's capabilities regarding allocation of IP-addresses there might be a problem joining certain networks. The discovery is taken care of by JXTA. The Peer-to-Peer architecture makes the system scalable too. It is only at setup that the configuration service have a big job to do if the peers are many, but the probability that one application will contain so many ActionBlocks that it will be to many to handle is not large. It is more probable that there are many applications operating at the same time and then the work for the configuration service will be spread.

Another thing that JXTA facilitates is the support for access to peers inside a firewall. Firewalls are a problem in distributed computing like the ActionBlocks system. But in JXTA it is easy to configure one peer inside and one outside the firewall to know about each other and to communicate via HTTP.

One request is that it shall be possible to temporary turn of an application. This is possible by the fact that the tag id specifies what is to be done. It is also possible to set a specific time in the tag so that the application will be turned of for that time and then start to operate again. All the data from the tag is obtained in the TagReaderSingelData document (Appendix I).

8.6.2 **Configuration**

As mentioned in section 8.2 only the setup is implemented in the configuration aspect. But some of the listed requirements are fulfilled though.

It is possible for several applications to operate at the same time and use the same transmitters. The configuration server coordinates the communication groups between the different applications so that a transmitter can communicate with several receivers at the same time. The ActionBlock may be allocated for a specific application. This is specified in the XML document (Appendix III). I wouldn’t, for example, appreciate if anyone else played his music on the loudspeaker that I was using at the time.

It is also possible to use several transmitters and one receiver and to combine the actions from two or more transmitters into one action. Each transmitter is
member in one communication group and the receiver listens on all the pipes and when the messages arrive they can be combined and depending on the combination a specific action can be made. This presumes that a message is sent when the tag are removed from the tag reader too.

The requirements dealing with the issue that the interaction designer shall be able to combine different ActionBlocks and logics in an easy way get its answers from the configuration manager and can just be described in theory. The ability for the interaction designer to combine different ActionBlocks in an easy way is to be done in the configuration appliance (Figure 21). The different ActionBlocks, logics etc are shown as icons in the interface and the software pieces are constructed as components with well defined interfaces between them which makes it possible for the interaction designer to combine different units with each other. Behind every icon there has to be a generic description of the component so that it is possible for the appliance to decide what is allowed or not. The application settings are assembled into one XML document, which is used at the setup later on.

8.6.3 Development

To add, remove and connect different ActionBlocks in the system is achieved by the interfaces between the event and the logic. As long as new sort of ActionBlocks have the same event module they can communicate with the others. The event module is the same for both transmitters and receiver and even for responders and the configuration service. This means that an ActionBlock may be both a transmitter and a receiver at the same time as long as the descriptive XML document defines the new type.

The question whether the system is easy to maintain and develop is discussed in chapter 10, but the intention is that it shall be. By using the same software for the event module in every ActionBlocks makes the system communication easier to survey for programmers that haven’t been working with the system before and when updating the same update may be done in all places. The programmer doesn’t have to do several different updates in different ActionBlocks. That will also facilitates the further development. The fact that all the logics are stored at one place makes it also easier for the programmer to survey what software there is available. The configuration service then takes care of the loading and the programmer doesn’t have to be concerned about were the different logics are to be installed. The component thinking might also make the programmer's work a bit easier because the different logics don’t have to be modified each time they are going to be used.
9  FURTHER WORK
The concepts have evolved out of different approaches toward ActionBlocks but there hasn’t been an investigation about how absolute the concepts have to be to satisfy the different roles. Is there room for some exceptions from the fundamental concepts? Such a study might be a good idea to perform. To make the architecture concepts really solid there also has to be made some more definitions of different occurrences and rules to make it really general and flexible.
There is also a lot of work left to make the prototype evolve to a complete application.

- The functionality has to be increased to also support different types of responders and appliances.
- A technical issue like firewalls restricts the usage of the system and to be able to force firewalls a relay peer has to be set up.
- The error control in the prototype has to be extended so that the ActionBlocks can find other ActionBlocks, which were part of the previous application, if the configuration service is out of operation.
- The configuration manager isn't implemented either and to make the system get the desired functionality this is an essential part to develop. It might be a good subject for another master thesis.
- The logic hasn't been discussed at all in chapter 8. There is an implementation of the demo described in the beginning of the chapter but how to build the different logic units in a uniform way have to be explored further.
- An XML database is required to store the different XML documents in the configuration service and manager.
- To make the application as easy to use as possible the ActionBlocks shall start by themselves when they are plugged into the network.
- Only one of the reconnection principles are implemented, the id principle. It is the principle that is most likely to be in use in this early stage, but while the project evolves the other reconnection principles has to be implemented. The location principle requires much additional software and hardware.
10 DISCUSSION

10.1 Comparisons with Existing Architectures

In section 3.2 I pointed out that not anyone of the presented architectures explicitly focused on all three interaction aspects. The proposed architecture for ActionBlocks does. But it is so different from the related architectures? What similarities are there and what differences? Let's look at each architecture once more.

Multiple Trivialities (3.1.1 Multiple Trivialities) is a project that also took place at the Space studio. The fundamental difference between the two architectures is that in Multiple Trivialities the ActionBlocks communicate via a server. It is the server that forwards all messages from the tag reader to the web server that displays the movements the user makes. In the ActionBlocks architecture the ActionBlocks themselves handle the communication during use.

Another difference is that all messages shared among ActionBlocks are formatted to XML document and the whole architecture relies on these documents for communication and for identification of different types of ActionBlocks. This makes the new architecture more flexible. In Multiple Trivialities all messages were sent as separate strings instead of clustered into one XML document. A similarity is the layered structure of the architecture. Through just modifying the connection to the interface IPP the applications can use the client-server architecture to communicate with each other.

Web Presence (3.1.2 Web Presence) is an extensive project that covers a lot of different situations. The main goal is to customize the environment to get the most out of it for the user's convenience. A compelling resemblance is the assumption that when the user has the intention to customize or configure the environment for his use it doesn't matter how the architecture look like. The essential is to be able to reach different services. Web Presence uses different types of architectures dependent on which situation the system is used in. A printer may act by itself on a request, but information where the bus is on its rout is obtained from a server (Kindberg et.al; 2000). The same view is mirrored in the configuration aspect. The difference is that the development aspect isn't a focal point at all in Web Presence, at least not explicit.

In the early stage of the work with the configuration service the Appliance Data Service (3.1.3 Appliance Data Service (ADS)) inspired me. In ADS, Huang et.al. also uses templates that describe what data is needed for a specific application. The template also identifies the services that is going to be invoked when a user connects his camera to an access point. These templates are XML documents and they have the same function as the XML document that defines what ActionBlocks that is required to start an ActionBlock application. Another resemblance is the role of what is called the Dataflow Manager and the role of the configuration service. The Dataflow manager is to "make certain an application has all the data it requires" (Huang et.al; 2001) the same task is performed by the configuration service. A significant difference between the two architectures is that the template has a role of minimizing the configuration of the device (Huang et.al; 2001) when the XML document in the ActionBlocks architecture is a foundation for the possibility to configure the system. In ADS the motto also is to keep the devices, that the user interacts with, simple and to centralize the software. The heaviest motivation for that is that it will relieve the user from using a PC to perform the task to unload the camera etc. and that maintenance will be easier (Huang et.al; 2001). Some similarity may be discerned though. To facilitate maintenance and further development the software (logic) also is kept centralized in the configuration service, but the individual ActionBlocks get the needed software when required which differ from the ADS approach.
Most of the ubiquitous applications involve three areas; natural interfaces, context-awareness and automated capture and access (Abowd, 1999) (Abowd & Mynatt, 2000) and so also in Interaction Spaces (3.1.4 Interaction Spaces), which is very focused on context awareness for interaction. This area is not the one that concerns the ActionBlocks system most. The context awareness in this case is limited to the location of the ActionBlocks. The rest is captured in the applications. Therefore the differences between the two architectures are more visual than the similarities. A resemblance to notice is that in Interaction Spaces a clean separation of concerns is practiced just like in the ActionBlocks architecture.

What is there to say about Tacoma (3.1.5 Tacoma)? The project is very different from the ActionBlock architecture, but there is one resemblance. In Tacoma the user write instructions or code in a SMS message and the code is then executed on a remote computer (Jacobsen & Johansen, 1998). A similar thing takes place at setup in the ActionBlocks architecture when the java source code is uploaded to the ActionBlocks. But in this case the programming is going on during the development aspect while I regard the Tacoma process as configuration.

The RFID Chef (3.1.6 RFID Chef) is an event-based system. An event is a message produced by an event source and sent to an event consumer (Langheinrich et.al, 2000). That can be compared with the messages between transmitters and receivers, but in RFID Chef the event handling is layered to takes care of some filtering, for example to avoid flickering screen when supplementary objects are added to the counter. In the RFID Chef every tagged item have an object representation within the system. In the ActionBlocks system the tagged item acts more like pointers even though they have a virtual representation.

In the tag configuration appliance the user can associate files to a specific tag (item) and in this way get a pointer to that data. The user may have the impression that the data is stored in the item. This is also just what Informative Thing (3.1.7 Informative Things) is about. But in Informative Things the idea is for the specific user to obtain the information "stored" in the item and own by him wherever he is. That isn't the issue in the ActionBlocks architecture were an application is a closed, definite system that can be used by anyone.

The resemblance between Invisible Interfaces (3.1.8 Invisible Interfaces) and the ActionBlocks architecture seems very striking at first sight. Invisible Interfaces work with RFID tags that are attached to physical objects that may be manipulated. Both architectures have an accent on physical representation that feels natural to the user (A wristwatch is a good representation for a calendar program, while both are occupied with time.). The ActionBlocks architecture addresses a broader area of appliance. Invisible Interfaces also has a shared database with tag ids and their mapped virtual counterparts. In the ActionBlocks system such database or responder can be divided into several that support different ActionBlocks and they can even be located at an ActionBlock.

Hive (3.1.9 Hive) is an interesting architecture that applies Peer-to-Peer architecture (Minar et.al, 1999) just like the ActionBlocks architecture in the use aspect. The most interesting thing is the GUI that makes a user able to compose a system of devices. That GUI is very similar to the configuration appliance that are described in section 7.3. It would have been very interesting to see Hive in use. Hive is founded on mobile agents that search for the resource and perform a task with it and communicate with other agents. In ActionBlocks the resource itself perform the task and communicates with other resources.

Three essential similarities may be discerned between Jini (3.1.10 JINI™) and the ActionBlocks system. Both focus on flexibility. Both use mobile code and they both have a form of lookup service or configuration service. They may work differently but they have the same purpose, to locate devices in the network. Jini was also a source of inspiration at the beginning of this work.
As we have seen the ActionBlocks architecture has similarities to all the presented architectures even though many differences also exists. In the article "Research Issues in Ubiquitous Computing" Alan J. Demers (Demers, 1994) discusses the difference between ubiquitous computing and Virtual Reality. Just like Weiser (Weiser, 1991) he states that they are the opposite of each other. Demers says, "The difference is not so much in the technical details as in the emphasis and goal." That sentence is as much applicable to the ActionBlocks architecture towards other architectures. The difference is in the rational behind the architectures. The ActionBlocks architecture is more conscious of different types of users. It is important to look through the eyes of the different user roles when constructing the architecture. In the ActionBlocks architecture the interaction views are the foundation of the concept. The similarities are founded in the fact that some issues are important in the ActionBlocks architecture as well as in the presented architectures. The same problems are dealt with, which sometimes result in similar solutions.

It is like in life. We are all different and we all strive for different goals in life, but now and then our paths cross. For a short time we share a sub goal and then our paths are parted again.

10.2 Reflections
To provide for ongoing ubiquitous computing research there are three software engineering issues to take care of. It is toolkit design, software structuring to separate different concerns and component integration (Abowd, 1999).

To make an ubiquitous application really good it has to evolve gradually and to make this evolution possible the software has to be structured in a way that separate different concerns in the application (Abowd, 1999). That is what is suggested in the ActionBlocks architecture but it can be carried out further during time. One can also say that the ActionBlocks architecture is a sort of toolkit were the interaction designer and the programmer has some pieces and rules to work with, to combine and to complement. The component integration is not implemented yet but the intention is that all the logic shall be a collection of components that can be united.

In theory the ActionBlocks architecture seams rather suitable for ubiquitous computing. The question is how the architecture will work in practice. Will it be interesting and natural for the user? Will it be the flexible toolkit that the interaction designer desires and will it be practical for the programmer?

The architecture is experimental and we have to remember that an architecture also has to evolve during time. In the next section I therefore discuss a set of issues to consider when making the next version. The issues are flexibility, some programming issues, implementation of the architecture, learning to use the system and customization.

10.2.1 Flexibility
To make it possible for the interaction designer to assemble systems that is appropriate for a specific interaction scenario there has to be ActionBlocks, logic etc. available to fit different situations. Such a library takes some time to build. Hopefully the flexible architecture will provide a foundation that makes this work easier and faster to achieve. But there will always be a risk that the different projects that will use ActionBlocks have very different demands and it will result in that the programmer has to make new components each time. Naturally this will be a rule in the beginning of the evolution of the architecture. But further on that won't be desirable more than once in a while because then some of the intention with the architecture will be lost.
10.2.2 Programming Issues
The programmer's role is to design ActionBlocks and in this work he has a need to
control the processes within the architecture. It may be awkward to him to just load
the logic to the configuration service and then leave it with that. It may be more
domesticated to put the logic at the right ActionBlocks directly. But doing so all
the logic that an ActionBlock can have has to be at place all the time. Otherwise it
isn't possible for the interaction designer to construct application by his own. That
would require computers with bigger capacity. In case of this procedure the
ActionBlocks should be able to locate each other all by them selves too and then
we talk about a system that would be much more expensive than the current. The
maintenance would be more demanding too.

An advantage (disregarding maintenance and the advantage to have all the logic
at one place) with the upload procedure is that the programmer doesn't have to
have physical ActionBlocks available when developing and testing the new
components. As pointed out in section 4.2 the programmer's view is that an
ActionBlock doesn't have to have a physical part. This view makes it possible for
the programmer to have software representations of the ActionBlocks while
developing.

At first it may seam that some programming issues, error handling for example,
will be easier when the user doesn't interact with the system with a mouse or a
keyboard. In the case of tag readers that might be true to a certain extent. But then
we have to consider other possible input devices such as a digital whiteboard and
speech recognition there the user's handwriting or voice has to be translated into a
digital representation. In such cases the error handling might be worse than in
ordinary programming. The advantages and disadvantages in ubiquitous and
ordinary programming respectively is probably a question of give and take.

10.2.3 Implementation
Of course there are some issues in the prototype that also can be discussed. Is
JXTA such a good choice for example? Some functionalities in the JXTA
implementation seam to relate to the fact that it usually is a user behind the
computer. A user that can make choices and decisions about which peer to contact,
react when a connection is lost and so on. In an ActionBlocks system there may not
even be an ordinary user interface. The ActionBlocks have to act on its own.

For example when an output pipe is created and the ActionBlock get a
connection with another ActionBlock that listens on that pipe, an event is created
and the sending ActionBlock get knowledge of the connection. The listener doesn't
get such an event. There are ways to implement this of course but it isn't automatic.
In the ActionBlocks architecture it is the listener that need the information about
the connection most. Such small details make some doubts arise, but JXTA offers
many advantages such as a platform that makes the communication issues totally
separated from the rest of the application and the XML base that corresponds well
with the ideas in the architecture. How far JXTA will take the architecture will be
established in the future.

10.2.4 Learning
In anthropological studies it has been established that interfaces that are discreet
are the best because the user ceases to think that they use technology and instead
they regard the technology as abilities they have themselves. The technology is so
embedded into their life that they take it for granted (Demers, 1994). To be able to
develop such applications with transparent interaction the user has to be able to
treat other forms of input devices in the same obvious way that mouse and
keyboard is handled today (Abowd, 1999). This requires some learning time. How
much time is the user ready to spend on learning, when the result is the same as
without technology? Most people wouldn't spend any time at all on learning such
thing. How many times haven't we heard that the computer saves time? But I say
not for a beginner. It is rather hard work to achieve a sufficient knowledge level to even be able to use Word in a way that really makes a significant advantage instead of writing by hand. I have many friends that have given up their tries, because they only need the computer occasionally and when they have to use the computer it consumes their time. For them is it for example much faster to write a letter by hand. To learn something the motivation has to be greater than the learning efforts. Harder to learn require higher motivation. The motivation is proportional to the added value in the new application. To make the application attractive to the user either the interface has to be so intuitive that the added value hasn't to be so high or the application have to be so exciting that the user just has to learn to use it. Weiser (Weiser, 1991) declare that the application is the whole point in ubiquitous computing. He says that it will be good applications that drive the development further. I am obliged to agree with him. An application has to add some extended value to features in the work that previously wasn't supported, not doing something over again in a different way. Only then the user starts using it continuously and only then new angles of incidents will occur.

10.2.5 Customization
In the discussion about flexibility, customization and configurability the question about what the user really shall see and do always arises. How much shall the user be allowed to manipulate the system? It is a kind of walking on a tight rope between making the system flexible enough to be interesting to the user and to make it easy to learn. The tag configuration manager, used by the user Jan in the scenario (section 1.1), and the tag configuration appliance is a step outside the original concept. The tag configuration appliance is meant for the interaction designer to attach tags to new ActionBlocks but the user can also use it to associate tags to different files. In this way the system get more flexible and adjusted to the user's needs. This use adds another level to the concept, a meta level. The user tailors the system by making choices. One choice may be that tag A will be associated to file A, but the user may change his mind later and associate the tag A to file B. The choices are contained in the meta-data. The tag configuration manager is an implementation of the meta-data approach (Lindeberg et.al, 2002), which means that the basic program is controlled by meta-data.

The tag configuration manager may also seem to blur the distinctions between the three roles, user, interaction designer and programmer. But then we have to remember that the roles may be contained in one person. The roles may slide into each other. The division into three roles is an abstraction. It is a comprehension of the intentions with an architecture, disregarding many details. Abstractions make it possible to think about important behaviors or issues without having to bother about the entire system (Kiczales, 1992). The abstraction into three roles is done to be able to focus on three different interaction views toward or relations to a computer system. In this case especially the ActionBlocks system. The question is if it is a relevant generalization. Does people in general have such approaches towards computer systems that I introduced in section 4.1 Approaches to ActionBlocks?

10.3 End of Discussion!
During the work with this thesis I have speculated in, as implied above, whether there really is a reason to adapt the architecture to different roles. The question got its answer one Sunday afternoon in the beginning of May when my brother came to visit my husband and me. It was really good motorcycle weather and my brother had bought a new voice controlled intercom for his motorcycle. My husband and

18 Abstraction - create a more general idea about something through disregarding details.
19 Generalization - Make a more general validity to something that was originally valid for just a few.
my brother discussed some problems with the microphone. When I entered into the
discussion I heard the following conversation:

H: But why don't you use the other microphone?
B: What microphone?
H: The microphone from your old intercom.
B: But this new one is voice controlled.
H: It doesn't matter. The voice control is in the box. The microphone hasn't
    anything to do with it.
B: Doesn't it!!?

My husband, my brother and I have very different interests and personalities. My
husband likes to construct and mend things, my brother isn't very interested in the
technical side of things but he likes to use technical devices and I am interested in
how the things can be used to make the existence easier. Really it is much more the
idea with the thing that appeal to me. Our interests mirror the way we look at for
example the intercom in the scenario above. My brother thinks that the microphone
has the ability to decide whether he talks or not and that the box makes the other
person hear what he says. For my husband all functionality is at one place in the
box because it's more convenient. For me the intercom is just a thing that makes it
possible to talk to each other when driving a motorcycle. Only when I enter the
role of constructor or mender will I question where the different functionalities
lies.

My husband's, my brother's and my interests correspond to the user’s,
interaction designer’s and the programmer’s views on the ActionBlocks
architecture. The conversation confirmed my opinion that the technology has to
adapt to the human and not the other way round. If it is possible to make the
architecture match the apprehension of the system for the three roles it ought to be
done. That is what it is all about "for each his own".
REFERENCES

Literature


Written Material

Barret, Rob & Maglio, Paul P: “Informative Things: How to attach information to the real world”. In UIST’98, San Francisco, CA.
Kiczales, Gregor et.al: "Aspect Oriented Programming”. In European Conference on Object-Oriented Programming (ECOOP), Finland, June 1997.
Lindeberg, Olle et.al.: "Using Metaobject Protocol to Implement Tailoring; Possibilities and Problems". In The 6th World Conference on Integrated Design & Process Technology (IDPT 2002), Pasadena, California, June 23-
BIBLIOGRAPHY

Literature


Written Material


Friday, Adrian et.al: “Supporting Service Discovery, Querying and Interaction In Ubiquitous Computing Environments”. In Second ACM International
workshop on Data engineering for wireless and mobile access, May 20, 2001, Santa Barbara, CA USA. Pp 7-13.


Internet Sources


<table>
<thead>
<tr>
<th>GLOSSARY</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>The counterpart of an event that take place in the real world.</td>
</tr>
<tr>
<td>ActionBlocks</td>
<td>A building block that consists of an intelligent part and a physical part.</td>
</tr>
<tr>
<td>Appliance</td>
<td>A program that assist the user in use of ActionBlocks.</td>
</tr>
<tr>
<td>Application</td>
<td>A specific combination of several ActionBlocks etc.</td>
</tr>
<tr>
<td>Configuration manager</td>
<td>A program that is used at system configuration.</td>
</tr>
<tr>
<td>Configuration service</td>
<td>A program that are used at setup.</td>
</tr>
<tr>
<td>Event</td>
<td>The counterpart of an action that take place in the digital world.</td>
</tr>
<tr>
<td>Logic</td>
<td>A piece of software in an ActionBlock.</td>
</tr>
<tr>
<td>Physical item</td>
<td>The part of the ActionBlock that the user can see and touch.</td>
</tr>
<tr>
<td>Receiver</td>
<td>An ActionBlock that receives data from other ActionBlocks and process it and expose it to the user.</td>
</tr>
<tr>
<td>Responder</td>
<td>A unit that responds to requests and doesn't have a physical part that the user interacts with.</td>
</tr>
<tr>
<td>RF network</td>
<td>Radio Frequency network</td>
</tr>
<tr>
<td>RFID</td>
<td>Radio Frequency Identification</td>
</tr>
<tr>
<td>Setup</td>
<td>The process to make the application come in start position. The ActionBlocks get knowledge of which other ActionBlocks to communicate with.</td>
</tr>
<tr>
<td>Symbolic representation</td>
<td>The part of the ActionBlock that holds the data and the functionality that is specific for the ActionBlock.</td>
</tr>
<tr>
<td>System configuration</td>
<td>The process when the interaction designer associate different ActionBlocks etc. to each other to create an application.</td>
</tr>
<tr>
<td>Tag</td>
<td>An integrated circuit that can hold a small amount of data that can be read by a tag reader.</td>
</tr>
<tr>
<td>Tag reader</td>
<td>A device that emit radio waves and can read data from tags.</td>
</tr>
<tr>
<td>TINI</td>
<td>A small web server.</td>
</tr>
<tr>
<td>Transmitter</td>
<td>An ActionBlock that sends data to other ActionBlocks.</td>
</tr>
<tr>
<td>Ubiquitous computing</td>
<td>Many computers that support many different users. Also known as disappearing computing.</td>
</tr>
</tbody>
</table>
APPENDIX I

TagReaderSingelData.xml

<?xml version="1.0" encoding="us-ascii"?>
<!DOCTYPE TagReaderSingleData [
<!ATTLIST TagReaderSingleData xmlns CDATA #FIXED
"http://space.interactiveinstitute.se/ipp/applications/TagReaderSing
eleData">
<!ELEMENT TagReaderSingleData (timestamp?,data+)>

<!ATTLIST data
cardid CDATA #IMPLIED
address CDATA #IMPLIED
value CDATA #REQUIRED
>

<!ATTLIST timestamp
time CDATA #REQUIRED
syntax CDATA #IMPLIED
>

<!ELEMENT data EMPTY>
<!ELEMENT timestamp EMPTY>
]>

<TagReaderSingleData>
  <timestamp time="2001-04-15 12:13:00" syntax="yyyy-mm-dd hh:mm:ss"/>
  <data cardid="0xF3" address="0x10" value="0x3"/>
</TagReaderSingleData>
APPENDIX II

ActionBlockData.xml

<?xml version="1.0" encoding="us-ascii"?>
<!DOCTYPE ActionBlockData [ 
<!ATTLIST ActionBlockData xmlns CDATA #FIXED>

<!ELEMENT ActionBlockData (data+)>
<!ATTLIST data homeGroup CDATA #REQUIRED
homePipe CDATA #REQUIRED
type CDATA #REQUIRED
elementName CDATA #REQUIRED
name CDATA #REQUIRED
actionBlock CDATA #REQUIRED
id CDATA #REQUIRED
location CDATA #REQUIRED
rendezvous CDATA #REQUIRED
> 

<!ELEMENT data EMPTY> ]>

<ActionBlockData>
  <data homeGroup="IPP_HomeGroup" homePipe="IPP_HomePipe"
       type="receiver" elementName="from_Configuration_service"
       name="Monitor_1"
       actionBlock="monitor" id="1" location="II_Malmo"
       rendezvous="CS"/>
</ActionBlockData>
APPENDIX III

SoundPlayer.xml

```xml
<?xml version="1.0" encoding="us-ascii"?>
<!DOCTYPE Application [ 
<!ATTLIST Application xmlns CDATA #FIXED >

<!ELEMENT Application (data+,ActionBlock+,software+)>

<!ATTLIST data
  name CDATA #REQUIRED
  id CDATA #REQUIRED
  priority1 CDATA #REQUIRED
  priority2 CDATA #REQUIRED
  priority3 CDATA #REQUIRED
  priority4 CDATA #REQUIRED
>

<!ATTLIST ActionBlock
  name CDATA #REQUIRED
  id CDATA #REQUIRED
  type CDATA #REQUIRED
  location CDATA #REQUIRED
  ab CDATA #REQUIRED
>

<!ATTLIST software
  name CDATA #REQUIRED
  id CDATA #REQUIRED
  ab CDATA #REQUIRED
>

<!ELEMENT data EMPTY>
<!ELEMENT ActionBlock EMPTY>
<!ELEMENT software EMPTY>
]>

<Application>
  <data name="SoundPlayer" id="1" priority1="ID" priority2="Logic"
        priority3="Location" priority4="ID_Location"/>
  <ActionBlock name="transmitterA" id="5" type="transmitter"
              ab="tagReader" location="" allocated="false" time="0"/>
  <ActionBlock name="loudspeaker1" id="6" type="receiver"
              ab="loudspeaker" location="II_Malmo" allocated="true"
              time="300000"/>
  <software name="SoundPlayer.java" id="1" ab="6"/>
</Application>
```
APPENDIX IV

ActiveApplications.xml

```xml
<?xml version="1.0" encoding="us-ascii"?>
<!DOCTYPE ActiveApplications [ 
<!ATTLIST ActiveApplications xmlns CDATA #FIXED >

<!ELEMENT ActiveApplications (data+)>

<!ATTLIST data 
   name CDATA #REQUIRED
   id CDATA #IMPLIED
>

<!ELEMENT data EMPTY>
]
>

<ActiveApplications>
   <data name="Puzzle" id="1"></data>
   <data name="Ruff" id="2"></data>
</ActiveApplications>
```
APPENDIX V

GroupData.xml

<?xml version="1.0" encoding="us-ascii"?>
<!DOCTYPE GroupData [ 
<!ATTLIST GroupData xmlns CDATA #FIXED >

<!ELEMENT GroupData (timestamp?,data+,addressee+)>

<!ATTLIST data
  groupName CDATA #REQUIRED
  pipeName CDATA #REQUIRED
  groupID CDATA #REQUIRED
  pipeID CDATA #REQUIRED
  elementName CDATA #REQUIRED
  rendezvous CDATA #REQUIRED
>

<!ATTLIST timestamp
  time CDATA #REQUIRED
  syntax CDATA #IMPLIED
>

<!ATTLIST addressee
  peerID CDATA #REQUIRED
>

<!ELEMENT data EMPTY>
<!ELEMENT addressee EMPTY>
<!ELEMENT timestamp EMPTY>
]

<GroupData>
  <timestamp time="2001-04-15 12:13:00" syntax="yyyy-mm-dd hh:mm:ss"/>
  <data groupName="aGroup" pipeName="aPipe" groupID="1" pipeID="1"
    elementName="hej" rendezvous="DS"/>
  <addressee peerID="2"/>
  <addressee peerID="5"/>
</GroupData>