RWTH AACHEN UNIVERSITY

The Patch Panel GUI

A Graphical Development Environment for Rapid Prototyping Interfaces for Ubicomp Environments

Diploma Thesis at the
Media Computing Group
Prof. Dr. Jan Borchers
Computer Science Department
RWTH Aachen University

by
René Reiners

Thesis advisor:
Prof. Dr. Jan Borchers

Second examiner:
Prof. Dr. Wolfgang Prinz

Registration date: Nov 04th, 2005
Submission date: May 29th, 2006

I hereby declare that I have created the work at hand completely on my own and used no other sources or tools than the ones listed. Citations were marked accordingly.

Hiermit versichere ich, die vorliegende Arbeit selbständig verfaßt und keine anderen als die angegebenen Quellen oder Hilfsmittel benutzt zu haben. Zitate wurden ordnungsgemäß gekennzeichnet.

Aachen, May 26, 2006

Contents

List of Figures

List of Tables

Abstract

Prototyping is an essential tool in the iterative design process following the DIA-cycle (Design-Implement-Analyze). After each implementation an evaluation should follow that suggests new improvements for the next version in the design process. The more design iterations are undertaken during the development process, the more the final release benefits from them.

Iterative design is only possible if prototypes can be developed easily, modified or even discarded without causing to many costs in time and money. Not every prototype will result in a final product but it assists in finding a solution that is well designed and will therefore in the end be accepted by the target group. Without prototyping, it may happen that a product is not suited to the requirements and results in a total failure.

Although there exist many tools that allow rapid prototyping in hardware and software fields, the support for prototyping in *ubiquitous environments* has room for improvement. In this kind of environment, several independent components communicate with each other and are controlled by central units. They are running in the background and therefore not used actively. With that concept of the "disappearing computer" first introduced by Weiser [1991] several problems occur ranging from hardware design to a supporting communication infrastructure.

The *iStuff project* which is under continuous development at the Media Computing Group, RWTH Aachen University Germany, addresses these problems and provides an infrastructure as well as a set of integrated components. Thus, the iStuff project allows the prototyping of ubiquitous scenarios, devices and services.

An intermediary service called the *Patch Panel* is used to specify the behavior of the iStuff components. When this work was started, the problem at hand was that user-friendly ways to configure different iStuff components and their behavior were missing. The Patch Panel was configured via a scripting language that is difficult to use for non-experts.

This thesis deals with the *development of a graphical user interface for the Patch Panel* that should replace the scripting language on the long run. This makes an easier use of the iStuff prototyping suite possible. Besides this key element of the work, other applications supporting the prototyping process were (re)designed. They are used in parallel to the Patch Panel and perform different necessary tasks. Among them is a wrapper that supports the easy management and configuration of the different iStuff components.

Überblick

Die Erstellung von Prototypen ist nach dem DIA-Zyklus (Design-Implement-Analyze) ein notwendiges Hilfsmittel im iterativen Design-Prozeß. Jeder Implementierung sollte eine Auswertung folgen, die neue Verbesserungen für die nächste Version im Entwurfsprozeß aufdeckt. Je mehr Design-Iterationen während der Entwicklung durchlaufen werden desto mehr profitiert das Endprodukt von ihnen.

Iteratives Design ist jedoch nur möglich, wenn Prototypen einfach erstellt, verändert oder gegebenenfalls wieder verworfen werden können, ohne hohe Verluste an Geld und Zeit zu verursachen. Nicht jeder Prototyp wird in einem fertigen Produkt enden, aber er kann dazu beitragen, eine gut entworfene Lösung zu finden, die am Ende von der Zielgruppe akzeptiert wird. Ohne die Erstellung von Protoypen kann es passieren, daß ein Produkt nicht den gestellten Anforderungen entspricht und somit zu einem Fehlschlag wird.

Obwohl bereits viele Werkzeuge zum schnellen Erstellen von Prototypen im Hardware- und Softwarebereich existieren, gibt es auf dem Gebiet der *ubiquitären Umgebungen* noch Raum für Verbesserungen. In dieser Art Umgebungen kommunizieren verschiedene unabhängige Komponenten, welche von zentralen Einheiten koordiniert und gesteuert werden. Sie arbeiten im Hintergrund und werden somit nicht aktiv verwendet. Mit diesem Konzept des "verschwindenden Computers", welches zuerst von Weiser [1991] vorgestellt wurde, entstehen neue Probleme, angefangen vom Entwurf passender Hardware bis hin zu einer unterstützenden Kommunikationsinfrastruktur.

Das *iStuff-Projekt*, welches sich unter fortlaufender Entwicklung am Lehrstuhl für Informatik X ("Media Computing Group") an der RWTH Aachen in Deutschland befindet, nimmt sich diesem Problem an und stellt eine solche Infrastruktur sowie eine Menge von integrierten Komponenten zur Verfügung. Somit erlaubt das iStuff-Projekt die Erstellung von Prototypen für ubiquitäre Szenarien, Geräte und Dienste.

Ein vermittelnder Dienst, genannt *Patch Panel*, wird verwendet, um das Verhalten der verschiedenen iStuff-Komponenten zu beschreiben. Als die vorliegende Arbeit begonnen wurde, bestand das Problem darin, benutzerfreundliche Wege zu finden, verschiedene iStuff Komponenten und deren Verhalten zu konfigurieren. Das Patch Panel wurde bisher mit Hilfe einer für Nicht-Experten schwer zu handhabenden Skriptsprache programmiert.

Diese Diplomarbeit beschreibt die *Entwicklung einer graphischen Benutzeroberfläche für das Patch Panel*, welche die Skriptsprache auf lange Sicht ersetzen soll. Damit soll eine einfachere Verwendung der iStuff Entwicklungsumgebung ermöglicht werden. Neben diesem Hauptelement der Arbeit wurden einige andere Anwendungen entwickelt bzw. angepaßt, um den Prozeß der Erstellung von Prototypen zu unterstützen. Sie werden parallel zum Patch Panel eingesetzt und erfüllen unterschiedliche Aufgaben. Es handelt sich unter anderem um eine Wrapper-Applikation, welche die einfache Verwaltung und Konfiguration der iStuff-Komponenten unterstützt.

Acknowledgements

From the beginning of this work and before, a lot of people supported me and helped me to overcome the usual obstacles that occur during the study progress. Some of them should be named here:

Many thanks go to my advisor, Rafael “Tico” Ballagas who always took time for meetings, constructive criticism and suggestions for the ongoing work. The final reviews were a very good support for me as well as the encouragement in times when it seemed hard to me to find solutions.

Professor Dr. Jan Borchers who always tries to keep a familial and collegial touch around the department. His support and feedback during the whole work made me feel comfortable, too.

Christoph Wilhelm, the department’s technician, who was always there when my computer and I were in trouble, respectively. With a lot of efforts and patience he always found a solution to current problems.

Eric Lee, David Holman and Daniel Spelmezan always had an open ear for questions concerning programming issues. Not all questions are always answered by Google and are seldom that well explained.

Eugen Yu, Faraz Ahmed Memon and Marius Wolf always provided good feedback and helped me a lot with setting up test scenarios and performing the user evaluation.

Not to forget Britta Grünberg who supported me with formal issues and hardware orderings.

My best friend Sebastian Kayser gave very valuable hints and feedback – thank you for reviewing my thesis.
Special thanks go to my girlfriend Beatrice Komischke who was always there for me, especially when I needed emotional support. She had to endure a couple of long and technical descriptions of my subject and always helped me to overcome the one or other contemporary crisis. Her feedback concerning my work also was a big assistance.

My parents, Renate and Heinz-Willy Reiners, who made it possible for me to study the subject of my choice in Aachen and always supported me concerning my private and educational life.
Without you, I would never have come that far.

Thank you!

Conventions

Throughout this thesis we use the following conventions:

Text conventions

Outlooks or remarks are set off in colored boxes.

OUTLOOK/REMARK:
Outlooks or remarks give additional information on a certain topic or provide ideas that can be applied to the described situation.

Definition:
Outlook/Remark

Source code and implementation symbols are written in typewriter-style text.

```
myClass
```

The whole thesis is written in American English.

Links to project sites or homepages of mentioned product and applications are shown in a footnote at the bottom of the appropriate page.

Chapter 1

Introduction

"Proper design can make a difference in our quality of life"

—Donald Norman in "The Design of Everyday Things"

Prototyping is one of the most important and powerful tools to get early insights and usability knowledge about a new product. It becomes clear very soon what deficits the product suffers from and what design decisions are well suited. Iterative design following the DIA-cycle (Design-Implement-Analyze) provides a very reliable basis for new concepts and product ideas. Nielsen [1993] describes this approach in more detail and states that if the iterative development principle is followed, the design matures with each iteration cycle.

The DIA-cycle design strategy is the key to successful development

Starting with concepts derived e.g. from brainstorming sessions and mind maps, storyboards or paper prototypes augment and help to communicate the initial idea. After the concepts are clarified, it should be possible to quickly implement low-fidelity, functional prototypes. It is important that after each iteration, the current version is evaluated. The evaluation results help to improve the design stepwise.

Concepts and low-fidelity prototypes

No matter what tools are used to create prototypes, the fi-

Iterations benefits

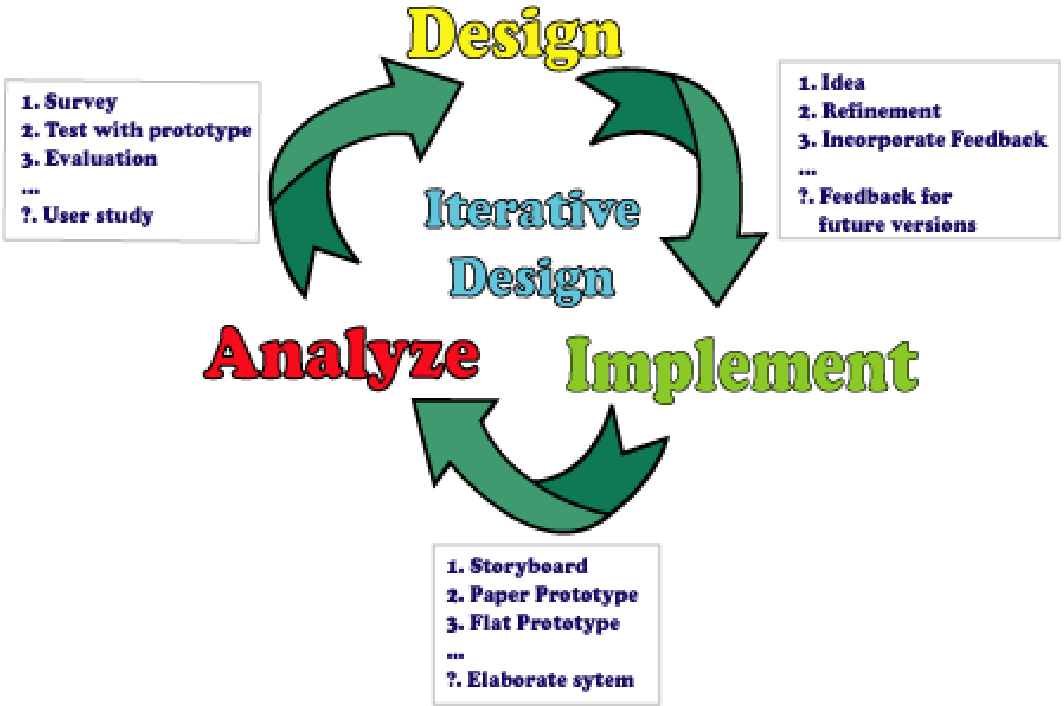


Figure 1.1: Following the DIA-Cycle in the development process ensures stepwise refinement of the design and continuous evaluation. With each iteration, the prototype becomes more elaborate and may even result in the final product.

nal result benefits from a lot of iterations where possible weaknesses and conceptual errors have certainly been discovered in early stages. So the probability of striking faults in the final product is decreased a lot.

Pre-evaluation is important for successful design

Developments that do not rely on prototyping and pre-evaluation techniques are more endangered to result in badly designed and therewith unusable products. For companies and research groups this could result in a considerable loss in time and money because the product would not be demanded by customers and therefore had to be improved and relaunched.

Worst case scenario

The redesign from the end point in a product's development cycle is mostly no longer profitable. Not to forget the damage left on the brand's label. This is why developers should argue for a design process following the DIA-cycle principle instead of the waterfall model, for example. As a worst case scenario, a originally innovative concept may result in a complete failure because of the wrong development strategy.

1.1 Prototyping in software and hardware

Standard software prototyping

For standard software prototyping, there is a variety of tools such as graphics software like Photo Shop[1] , Adobe Flash[2] or presentation software (e.g. Microsoft PowerPoint[3] or Apple Keynote[4] , included in the iWork software package). Other integrated development environments provide ways to rapidly construct a first version of the application's user interface. Apple Interface Builder[5] , Borland JBuilder[6] or Borland Delphi[7] are famous representatives of that class of development environments. With the latter ones, reuse is also provided as the developer can use the created interface and start implementing the underlying functionality.

Physical models

Hardware prototyping manifests itself in shape studies (e.g. aerodynamic issues in automotive or aircraft design), or usability studies analyzing the comfort to use a certain device all day. An example is the development of the Palm Handheld, where the development team around Jeff Hawkins carried a piece of wood that corresponded to the planned shape and weight of the future product (cf. Obendorf [2005], Bergman and Haitani [2000] and Butter and Pogue [2002]). From the evaluation of that early prototype important information on physical limits of the planned device could be derived. Aesthetic issues can also be analyzed by asking potential customers. In later iterations of

[1] http://www.adobe.com/digitalimag/main.html
[2] http://www.macromedia.com/software/flash/flashpro/
[3] http://office.microsoft.com/en-us/FX010857971033.aspx
[4] http://www.apple.com/iwork/keynote/
[5] http://developer.apple.com/tools/interfacebuilder.html
[6] http://www.borland.com/us/products/jbuilder/index.html
[7] http://www.borland.com/us/products/delphi/index.html

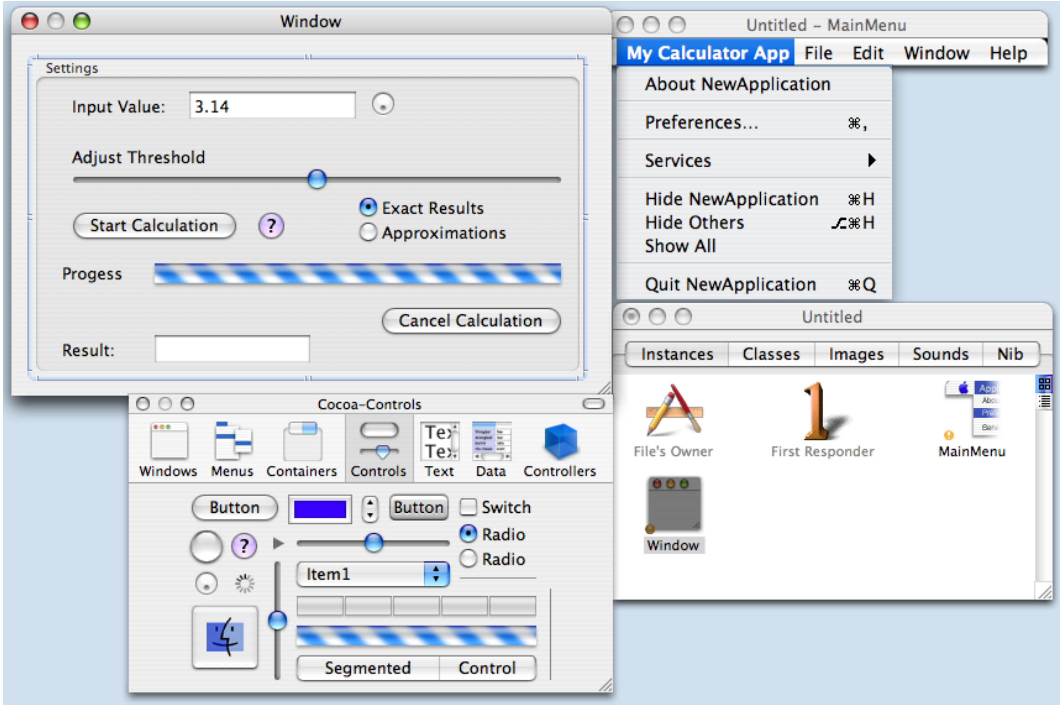


Figure 1.2: Tools like Apple's Interface Builder make it possible to quickly arrange UI elements and gain a first impression of the result. The design can be reused as the missing functionality can be added in later iterations.

the design, prototypes can also be used to directly test all the named issues and to create usability studies.

Rapid prototyping

Summarizing the above paragraphs one can say that prototyping is extremely important for successful design. In the "traditional" fields of hardware and software, prototyping more and more enters the designers' minds. Rapid prototyping allows even faster creation of concepts and iterations can be performed more easily; They should even be less time consuming and cost intensive. Many iterations follow each other and provide a very stable development basis.

1.2 Merging both fields

HW / SW Combination

For the software and hardware prototyping fields seen separately many proven methods exists. It becomes difficult when both parties should be joined. New design ideas that augment existing ones or introduce something completely innovative lacks from the possibility to quickly combine the ingredients. For standard software, horizontal prototypes can be created with the help of UI tools as mentioned, completely new ideas can be visualized with different graphical applications. Even a feel can be communicated by simulating the behavior of the software. In the hardware field it is similar: Model-creation is widely known as a help for conducting first analysis.

Problems with different parts

When the two fields are combined, however, the problem occurs that multiple different hardware parts have to be interpreted by a semantic unit that is provided by software. After processing the input, the software must be able to control the same or other hardware parts.

A design scenario

To motivate this situation with an example imagine the following scenario:
Bob is part of a design group that explores novel mobile phone interactions. One day, he is asked to augment a standard mobile phone to become situation aware. For this purpose, he equips the device with small sensors that provide information of the environment around it and its state (e.g. whether it is held, pocketed or lying somewhere). The sensors are capable of detecting light, vibration, noise or pressure whereas many other meaningful measurements are imaginable.

Augmentation of mobile phones

In order to examine the impacts on the interaction with the mobile phone, Bob runs into a number of problems; The sensors could indeed be physically attached to the mobile phone in a easy way but where should the received data be processed? The sensors are delivered from different manufacturers and they all use different data formats. Even if he finds a way to establish a communication among the different parts of the setup (e.g. by soldering the hardware parts), the mobile phone had to be reprogrammed in order to be able to manipulate its behavior.

High costs

Bob knows that these steps are manageable in order to create a rough looking prototype, but it would obviously be very time-consuming. Besides that, the time consumption and costs for the reprogramming work would increase dramatically. He decides that this is far too complex and expensive for a design task that should elaborate the general applicability of a concept. Even if it was accepted, software changes would again consume a lot of time and money. For Bob and his design crew, such an approach would completely contradict to the idea of rapid prototyping. Changes and refinements cannot be performed in a productive way.

Ubiquitous computing

The above scenario applies to prototyping in the "classical" fields as well as to the area of *ubiquitous computing*. In this research field initiated by Marc Weiser in his work "The Computer of the 21st Century" (cf. Weiser [1991]) smart devices equipped with sensors or capable of providing information should silently integrate into the environment. The communication between different devices should ideally take place in a wireless manner. The current situation where a personal computer drags all the attention towards itself should completely be inverted such that users are able to concentrate on the tasks they wish to perform instead of caring about the interaction. Computations can be performed inside the smart devices themselves or performed on machines inside the room. Connection and tasks management must not be the user's concern.

Ubiquitous environments

All these gadget systems together built up an *ubiquitous environment*. Weiser compares his idea to the ancient art of writing. Nowadays we consume and provide information by simply reading or writing it - we are making use of this technique although we do not mandatorily need to know how to produce ink or paper, for example. Other examples where techniques have become ubiquitous are radios or motors and engines in a car. Radios are built in a lot of environments and the usage of them is mostly unconscious (at least the fact that we know *what* a radio is good for, not necessarily the way *how* to use all the features it provides). In a car, we are surrounded by a large amount of motors we are not aware of while using them: In electrical windows, air-conditioning, steering-assistance, etc.

Living in ubiquitous environments

This concept of working with technique without having to know much about the details of the underlying structure is the core idea of ubiquitous computing. "Working" also means "using" or even "living" in ubicomp environments.

Only few ubicomp applications

Enabling the developer to rapidly prototype interactions in ubicomp environments and conventional design tasks is mandatory for making use of the DIA-cycle design principle. Although the concept introduced by Marc Weiser is already over 15 years old, only few examples of ubiquitous computing applications have made it to market. One part of the problem may be that in order to design a ubiquitous computing application expertise in hardware, networking and embedded systems programming is needed.

The iStuff project and graphical support

The goal of the *iStuff project* (cf. chapter 2) is to make prototyping ubiquitous computing applications accessible to interaction designers and to help increase the pace of innovation in this kind of environments. The thesis at hand supports this concept in presenting a new graphical support for the design process and a way that abstracts from direct programming tasks.

1.3 Thesis structure

GUI support for developers

As three different but yet related prototyping areas are outlined, the fundamental problem this work deals with is pointed out: With an existing communication infrastructure as a foundation for the exploration of new design spaces based upon work done by Card et al. [1990] and Buxton [1983] , for example. An effective way to enable the designer to quickly combine hardware as well as software components and configure them without much programming and constructing effort needs to be found. A graphical user interface that applies a data-flow metaphor seems to point to the right direction. The usefulness and the users' acceptance of a GUI designed for the prototyping tasks as described is analyzed in this work. The infrastructure yet to be explained can be used in the field of prototyping that combines software and hardware aspects in the design process as well as the ubiquitous computing research area.

Prototyping support for the iStuff project

Although many approaches are available for software or traditional hardware prototyping, tools supporting the prototyping process as defined above are still under exploration. Some approaches are oriented at the end user, others follow different ideas. The prototyping support presented here concentrates on the iStuff project (cf. chapter 2) which is under constant development at the Media Computing Group at RWTH Aachen, Germany.

Chapter 2: The iStuff project in detail

The following chapter describes the iStuff project in more detail as it constitutes the foundation of the communication infrastructure which is based on a publish/subscribe mechanism in a tuple space.

Chapter 3: Related work

Chapter three discusses related work and points out what features were well-suited or missing in existing approaches when this work was started and thus justifies the development of a custom tool.

Chapter 4: GUI prototyping

In the fourth chapter custom design concepts are presented as well as paper prototypes that show the evolution of the initial idea of a prototyping GUI. A summary of new concepts and those found during the survey is presented.

Chapter 5: Apple Quartz Composer as a basis

The next step follows in terms of a very close look at Apples latest graphical development application, Quartz Composer[8] , that encourages to leave the path of completely developing a new graphical user interface and to modify an existing and very useful technique. At the end of this chapter, already implemented concepts and ideas that had to be added are presented.

Chapter 6: Quartz Composer modification

Chapter six describes the complete iStuff modification of the Quartz Composer and its architecture. The possibilities to further extend this modification is explained in detail. Implemented examples found in literature and projects performed by the Media Computing Group are presented at the end of this chapter. The replication of existing implementations from the literature should underline the versatility of the devised tool.

Chapter 7: Evaluation

To evaluate the design of the prototyping application a user study was performed. Its results are presented in chapter seven. A representative group of graduate students was introduced to the iStuff project and asked to accomplish certain prototyping tasks. The study compares the new graph-

[8]http://www.developer.apple.com/quartzcomposer

ical application with the existing scripting language. Factors like the tool's acceptance and the interest in future extensions were also analyzed in terms of Likert-scales.

Chapter 8: Summary and outlook

The final chapter summarizes the results of this work. An outlook and issues that are still open and interesting aspects of future work are given. The ongoing development of the entire iStuff project can be tracked at the project's site[9] .

[9] http://media.informatik.rwth-aachen.de/istuff/

Chapter 2

The iStuff project

iROS basis

The goal of the iStuff project (Ballagas et al. [2003]) is to simplify the exploration of alternative interaction techniques. It includes a toolkit of physical devices and a flexible software infrastructure based on the *Interactive Room Operating System (iROS)* (cf. Johanson et al. [2002]). The focus lies on the impacts of interaction changes rising from leaving the conventional desktop metaphor and exploring the field of interactive environments where all components work together and should be reconfigurable at runtime (cf. Borchers et al. [2002]). So input control can be shifted to different or multiple output devices in realtime. The underlying iROS platform provides mechanisms allowing the connection of the devices and services in a local network over that the complete communication among all included devices takes place.

Integration of mobile devices

Part of this platform is the *Event Heap* (cf. Johanson and Fox [2002]) that establishes the communication between components via the local network. Participating components can connect to the Event Heap to post information in form of events to it or register for certain events and react to them as soon as they appear.

Intermediation

Since the information is posted by one communication member is not necessarily meaningful for others, an intermediary service called the *Patch Panel* (cf. Ballagas [2004]) runs on top of the Event Heap. The Patch Panel adopts the information provided by different components and maps it to the needs of other members of the network. That way, mappings can be defined that enable different components to provide and consume information.

Extension of iStuff components

A lot of iStuff components (cf. Ballagas et al. [2003]), enable developers and researchers to rapidly integrate existing and newly developed devices into the project. As the project is under continuous development, its scope was extended to be also applied to other fields than ubiquitous computing. Prototyping for mobile phone interaction and their augmentation (e.g. with new kinds of sensors), for example, has been realized in the *iStuff Mobile* work, performed by Memon [2006]. With the iStuff toolkit, rapid hardware prototyping becomes possible because the infrastructure provides a level of indirection so that different parts do not have to be directly connected or compatible.

Events have a *descriptive* character

In contrast to different work like the Speakeasy approach developed by Newman et al. [2002] and also described in Edwards et al. [2002], where a direct data-driven communication is established among different components, the mechanism used in the iStuff project, has a descriptive character. There, the *behavior* of the components is described and only atomic information between them, i.e. number, string and boolean values, is exchanged. This way, the Event Heap and the Patch Panel focus on control flow.

Easy recombination

Technical aspects are implemented inside of proxies and are encapsulated into an iStuff entity. The descriptive abstraction allows very easy recombination of components and reconfiguration of their behavior by changing the commands sent to the devices' proxies. Thus, new concepts of physical user interfaces can be explored and innovative applications derived.

2.1 Currently integrated components

Off-the-shelf hardware

The iStuff toolkit is constantly extended with different kinds of hard- and software that serve as input and output components, respectively. As each component seen separately only provides little functionality, the components can be recombined among each other and help to augment other devices currently not integrated into the toolkit in order to explore new functionalities. Instead of exclusively building custom hardware, standard off-the-shelf components are augmented with technology that enables them to connect to the local network.

Variety of components

A list of currently integrated devices reaches from self-built components like the iSlider, the iDog, iPens or iButtons to off-the-shelf sensor kits that can be connected wirelessly, via Bluetooth or USB, respectively, to a computer (Ballagas et al. [2003]). They should provide a JAVA or C(+/++) - API to facilitate the integration into the toolkit which is a matter of hours as the integration principle always is the same. Figure 2.1 shows some of the first iStuff components.

Figure 2.1: Some of the first iStuff toolkit components that can be connected wirelessly or via Bluetooth or USB.

List of currently integrated components

At the moment, the following devices and kits are integrated (Additional information about the functionality of the components can be retrieved from the products' homepages):

- "Traditional iStuff components"[1] like iButtons, iSliders or the iDog
- Phidgets[2]
- Teleo[3]

[1]http://media.informatik.rwth-aachen.de/istuff/
[2]http://phidgets.com
[3]http://teleo.com

- SmartIts[4]
- BlueSentry[5]
- Nokia Series 60[6] mobile phones as part of the iStuff Mobile work (Ballagas et al. [2006b])
- Software controllers for Microsoft Powerpoint or Keynote presentations
- Triggers for the execution of AppleScripts
- Integration of spoken commands using speech recognition software
- several other helpful hard and software tools

Download package

The developers package is available for download on the berlios developer site[7] . Examples will be presented in the further proceeding of this thesis, when several interaction scenarios from literature are rebuilt in order to show the potential of the iStuff in combination with the newly developed Patch Panel interface.

Additional information

The following sections describe the underlying architecture the iStuff components make use of. More information as well as tutorials can be found on the iStuff project homepage.

2.2 iROS communication structure

The ancestor of the iStuff project is the iROS work at Stanford University

Johanson et al. [2002] explore in their work "The Interactive Workspace Project" issues of human-computer interaction. For their experiments they integrated several interactive devices into an interactive room like a large display device that allowed pen interaction, three touch-sensitive

[4]http://smartits.com
[5]http://bluesentry.com
[6]http://www.nokia.com
[7]http://developer.berlios.de/projects/istuff

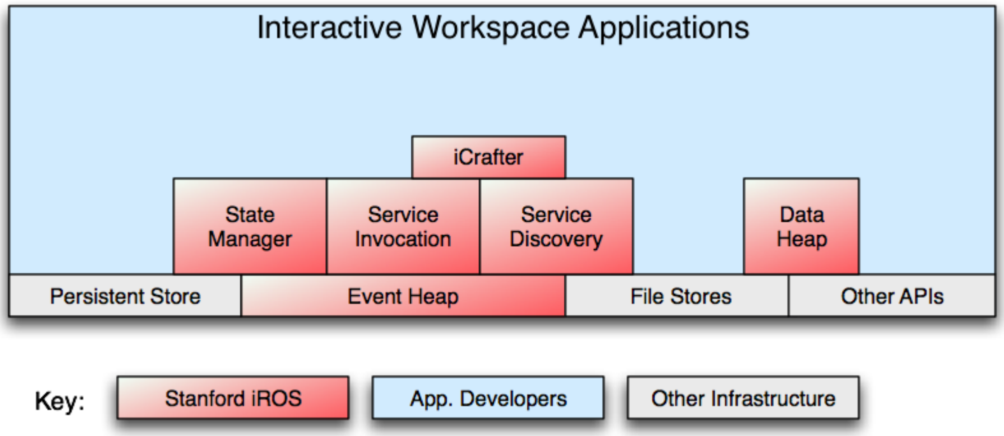


Figure 2.2: The different iROS components working together. Detailed information can be found in Johanson et al. [2002].

white-board sized displays arranged in a row and a conference room table with a built in display. Additional equipment of the room consisted of cameras, microphones, wireless LAN support and a variety of wireless buttons. With that setup, several usage modalities like moving data, moving control or dynamic application coordination should be explored in the interactive environment.

iROS architecture

The underlying architecture that allowed the interoperation of all components in the room was the *Interactive Room Operating System (iROS)* that consists of three sub-systems: The *Data Heap*, the *iCrafter* and the *Event Heap*. The Data Heap and the iCrafter were built in order to realize the concepts of moving data and control between different devices, respectively. The functionality of these two components can be studied in further details in the cited article. The only component that necessarily had to be used by an iROS component is the Event Heap because it is the underlying communication infrastructure for applications, services and devices within the interactive workspace. Figure 2.2 shows the principal organization of the iROS system.

2.2.1 Event Heap

The Event Heap as an information repository

The Event Heap offers decoupling communicating devices and applications so far from each other, that a possible failure of one party does not affect others. *Publish-subscribe-semantics* are used to achieve a cooperation in which the different participants are not directly dependent on each other. The Event Heap provides a central repository to which all parties can connect, post and retrieve information from. The data passed is encapsulated in a data structure called *event* for the rest of this thesis.

Events are collections of key-value pairs

An event consists of a collection of an arbitrary number of key-value fields. Some fields like "Event Name", "Time-To-Live, "SenderID" or "Creation Time" are standard fields included in every event. Application-specific fields can individually be added in order to provide special information. They could be created based on number values read by sensors or strings an application wants to send, for example. Only atomic information based on double, string or boolean values is exchanged via the Event Heap. A conceptual illustration of an event generated by a key press is shown in figure 2.3.

Fast communication

Events have a descriptive character to be interpreted by the receiving application. This allows very fast communication and avoids network congestion as the data packages remain small.

Pattern matching

Filtering out events for specific listeners is possible by by comparing them to patterns specified by the receiver. Thus, only events that can be processed by the receiver are read. With that approach, every participant simply fires events to the Event Heap and does not have to care whether they are consumed or not. Listeners wait for the matching pattern to appear on the Event Heap and read it.

Time to live

Events that are not consumed disband after a specified time limit. A standard TCP/IP protocol and several APIs including C++ and JAVA make it easy to implement clients subscribing to the Event Heap, post and consume events as illustrated in figure 2.4. More detailed information can be found in work performed by Johanson and Fox [2002] or on

Event Type	*TextEvent*
Character	*97*
ProxyID	*"localhost"*
TimeStamp	*1146580147486*
Source App.	*"TextEventEngine"*
TimeToLive	*500*

Figure 2.3: An excerpt of fields that are encapsulated inside an event. Some are standard fields, others like "character" contain individual data (an ASCII code in this case). The number of fields is arbitrary.

the Stanford University website[8] .

Interaction between arbitrary components

Applying the mechanism described, every arbitrary application or device connectable to the network is enabled to interact with any other party by using the Event Heap. That way, communication is established although the different participants were not designed to cooperate or communicate with each other.

Proxy strategy

In case that a component is not capable of connecting to the local network, a proxy strategy is applied. A proxy runs on a computer connected to the Event Heap and encapsulates the data it receives from the device into an event. Received events are interpreted to controls the attached device via USB, Bluetooth or wirelessly. Figure 2.5 illustrates the relationships described.

[8]http://iwork.stanford.edu/docs/eheap/index.html

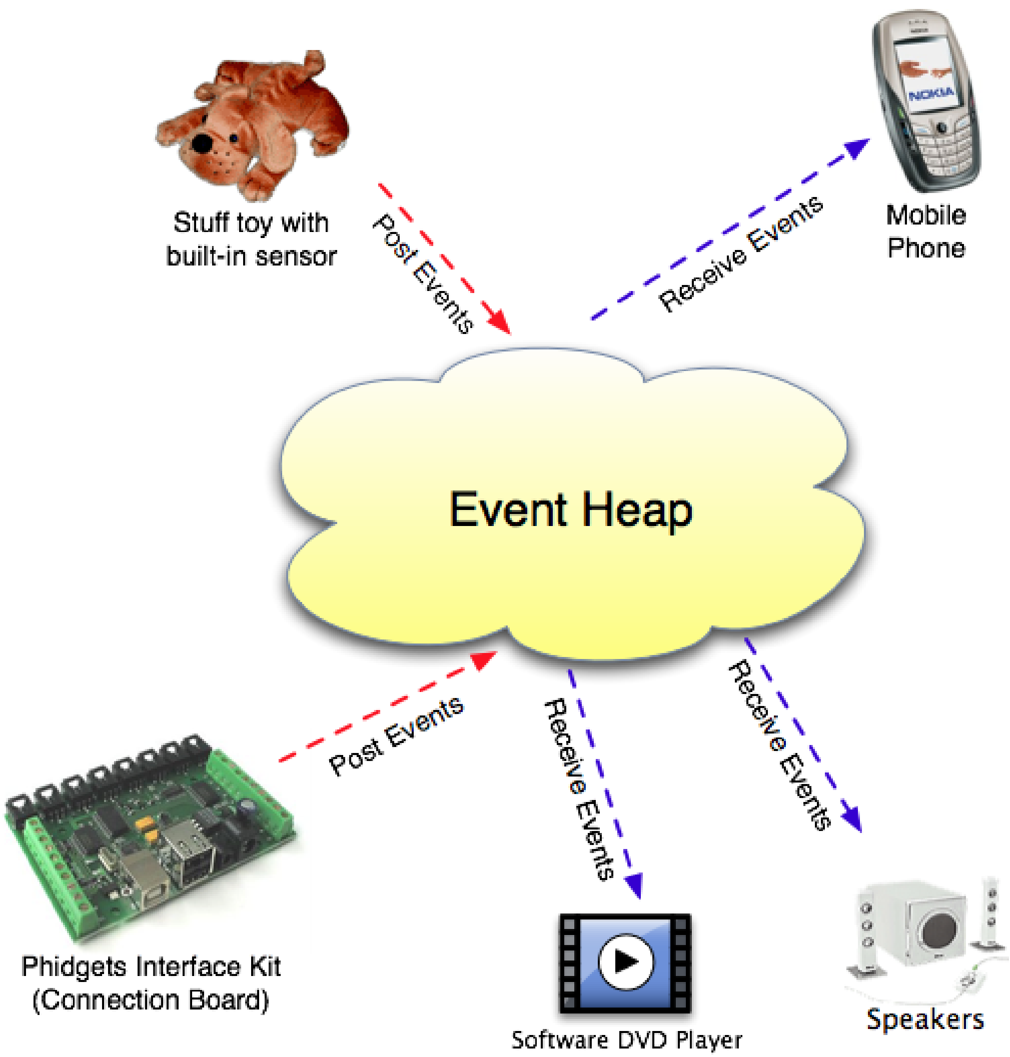


Figure 2.4: An illustration of the Event Heap with different devices connected. Although the components were not designed to interact with each other, this actually becomes possible with the iROS infrastructure.

2.2.2 Patch Panel

Different iStuff components use different events

After the communication structure is set, the problem rising now is that events of different types are posted to the Event Heap that are not necessarily interpretable by other proxies. Of course, senders could be hardcoded to post certain events types that would be interpretable, but then all benefits like the dynamic reconfiguration of the relation-

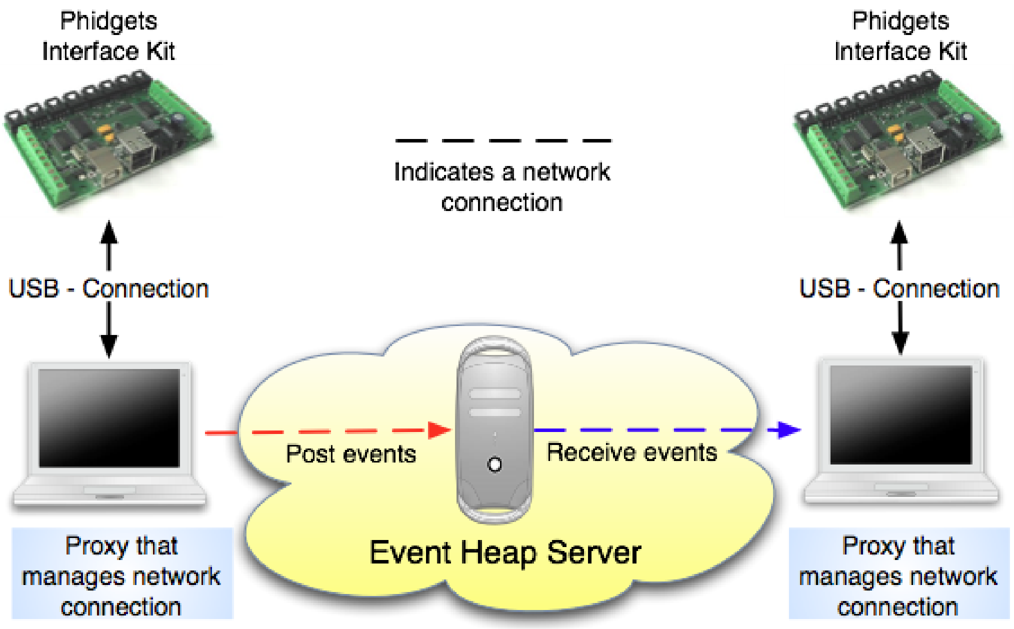


Figure 2.5: The control and connection management of devices like sensors is controlled by a software proxy running on a computer. The device is connected to this machine directly (via USB, Bluetooth, etc.) and so indirectly with the Event Heap. The proxy is responsible for the event handling and the interpretation of received events.

ship between different components and therewith the flexibility of the approach would be lost. Each proxy had to be reprogrammed before each run what would mean that the reusability would not be give anymore.

The Patch Panel intermediary service maps events

The solution to the problem is the *Patch Panel*, an intermediary service that runs on top of the Event Heap and that can be configured at runtime in order to register for certain events, consume them from the Event Heap and post new events such that they correspond to the format the receivers are expecting. The intermediation is reconfigurable at runtime and allows the establishment of communication between applications that can not work with each other. The flexibility of the structure is retained because mappings can be changed at runtime or events can be multiplied in order to be received by several (different) consumers. The values transmitted inside events can be transformed mathematically, casted to other types or left out completely. This all can be specified with the Patch Panel.

Interaction example

The following example which is also shown in figure 2.6 should clarify this method.

Used components

The iDog is a soft toy augmented with an accelerometer that detects whether the toy is moved and a WiFi chip that allows the connection to the local network. The chip's proxy is programmed to connect to the Event Heap and post events of type "iDog" with a field "Force" that contains the value of the sensor readings. Somewhere else in the interactive room, standard speakers are connected to a computer's sound card. A software proxy running on this machine subscribes to events of type "iSpeakers".

Mapping of two event types

Furthermore, the proxy expects a string field inside the event which specifies a path to a WAVE-file to be played by the proxy. The task of the patch panel is now to register for events of type "iDog", read them from the Event Heap and post a new event of type "iSpeakers" with a field containing a predefined filename.

Establishment of communication

The iDog proxy simply fires its values to the Event Heap and forgets about them. The iSpeakers wait for an appropriate event on the Event Heap which is delivered by the Patch Panel. So the communication between two completely different applications is established. In the described scenario, the iDog "barks" through the speakers standing in the room when it is lifted.

Patch Panel mappings

The Patch Panel not only provides 1:1, 1:n and m:1 mappings. It is flexible enough to receive one event and notify several other "observers" by posting new events with the appropriate formats to the Event Heap. It could also register for a number of events and map them to one single event, suitable for only one application.

The Patch Panel also implements state machines

Another mighty mechanism supports the Patch Panel's flexibility: The capability of implementing state machines. Consider a toggle button as an example; With an iButton the lights in the room should be controlled. But as the button is stateless, the Patch Panel has to define states it is currently in. The setup would be constructed as shown in figure 2.7. Now, with each button event, the Patch Panel re-

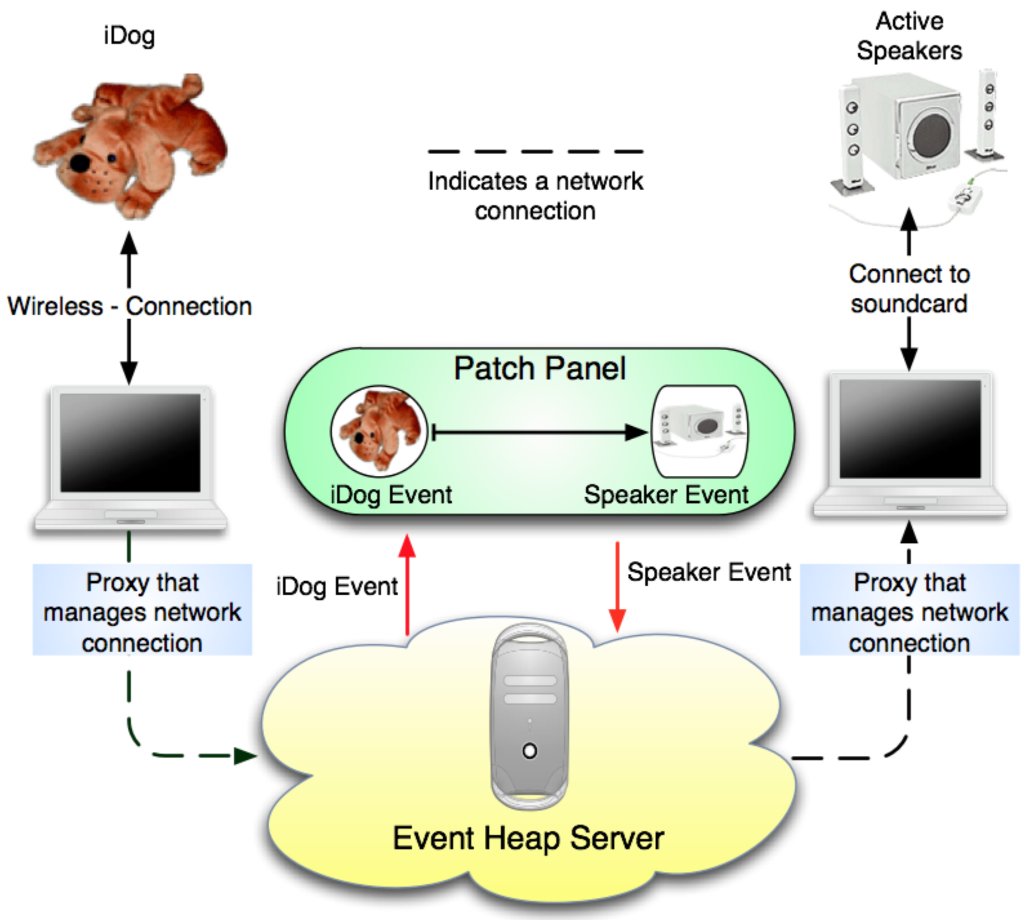


Figure 2.6: The Patch Panel as an intermediary service maps events of different types. Inside the Patch Panel, transformation and multiplication of events can also take place.

configures itself and therewith implements a state machine. At the first press it posts events to turn on the lights, then it reconfigures itself to post events that contain the information that the lights should turn off. The state machines implemented can become arbitrarily complex.

2.2.3 Configuring the Patch Panel so far

Scripting language

The configuration of the Patch Panel at the time this thesis was started took place by using a scripting language. Map-

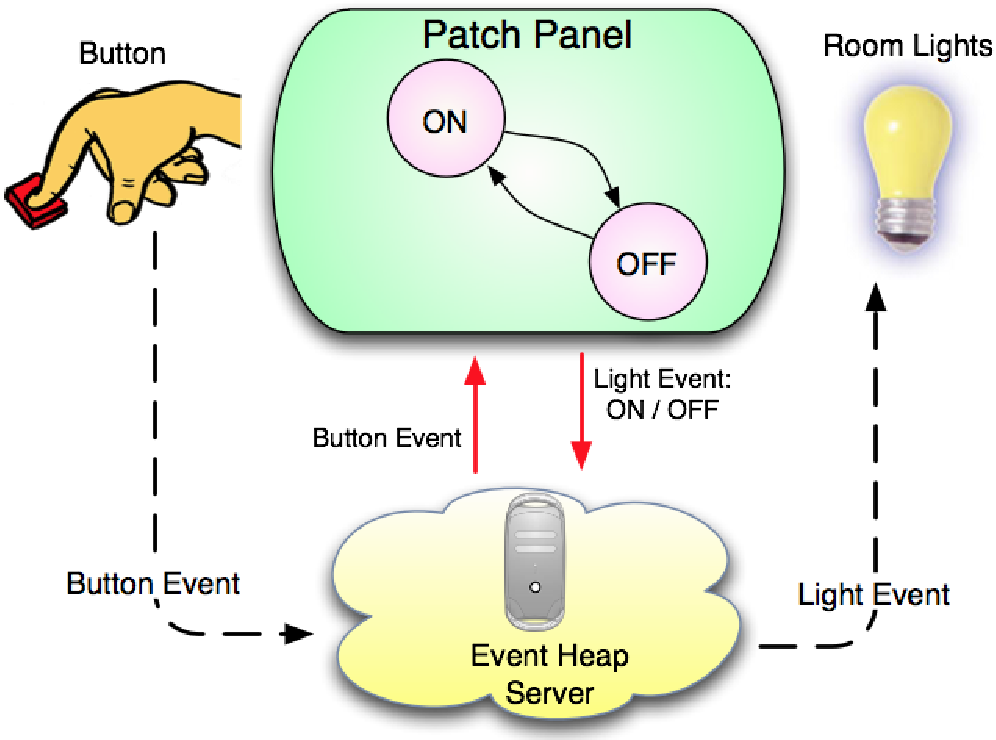


Figure 2.7: The Patch Panel implements arbitrary complex state machines. Here with every button event the state is switched and the corresponding light event is sent in order to control the room lights.

pings and state transitions were implemented in files. The listing below shows the implementation of the toggle button example described in the last section. Once an event to turn on the room lights was sent, the state is changed such that on the next button press, an event to turn off the lights is sent.

Script code

The code for the described script reads as follows:

```
state Off {
  on Button(id=red); {
     Lights(brightness = 10);
     Projector(powerOn=true);
     goto On;
  }
}
```

```
state On {
  on Button(id=red) {
  Lights(brightness = 0);
  Projector(powerOn=false);
  goto Off;
  }
}
```

The scripting language was also extended to a very sophisticated degree by Yu [2006] but it still forces the developer to programmatically define mappings and transitions.

Programmatic specifications

A quite basic version of a Patch Panel GUI was also implemented in JAVA (cf. Figure 2.8), but their capabilities were very limited in terms of usability and reconfigurability.

Former basic version of a Patch Panel GUI

The "beginners-mode" only allowed a very small number of mappings to be defined, the "advanced-mode" basically allowed a tree visualization of written script files. It is not necessary to mention that this way of configuring the Patch Panel, though open and flexible, is very time consuming and not suited to rapid prototyping. Not only because of the time but also because of lacking possibilities of quickly changing mappings and behavior at runtime.

Different modes

A look at that part of the iStuff project clearly states out that for rapid prototyping, a new way of implementing the Patch Panel had to be found. A graphical user interface seemed to be the correct approach to integrate "liveness" of the prototyping environment.

Need of a new Patch Panel implementation

This is not given by the scripting language and the described version of the Patch Panel GUI; Changes in the scripts need to be recompiled before taken into effect. It is also difficult to modify parameters at runtime for testing purposes. The basic GUI also did not allow this degree of flexibility, neither in the "beginners-mode", where only basic mappings could be defined, nor in the advanced mode that only loads already compiled scripts.

Liveness of changes not supported

The next chapter analyzes existing concepts and approaches for supporting hardware prototyping with graphical assistance. Their benefits and lacks are listed and com-

The following chapter compares related work

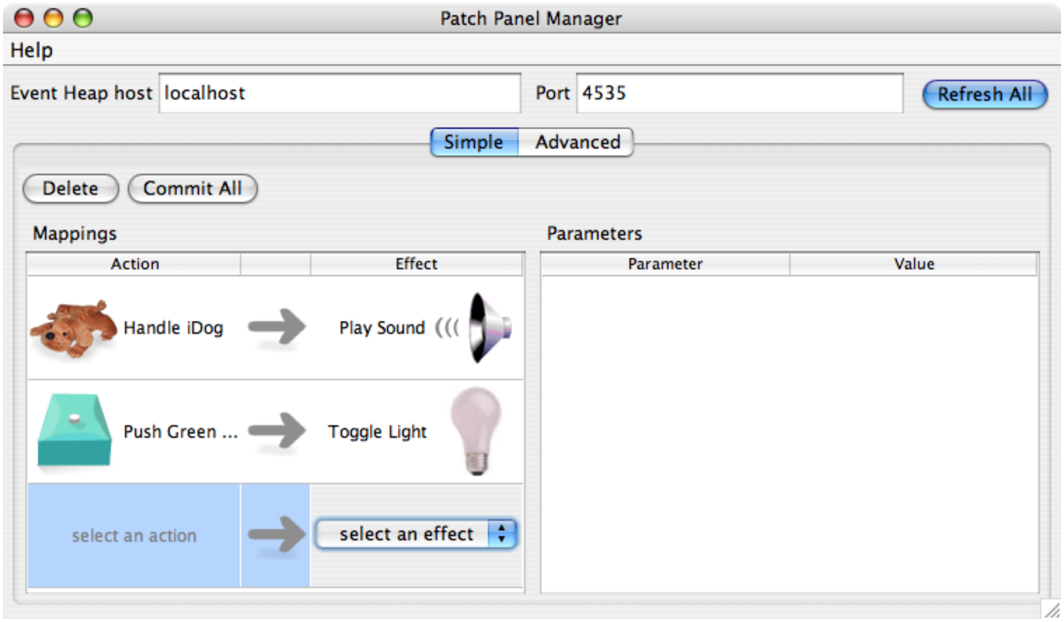


Figure 2.8: Screenshot of the existing GUI at the time when this thesis was begun.

pared to the needs for a new Patch Panel GUI.

Needed features in the new approach

Key demands to that GUI are the currently missing "liveness" of the prototyping environment such that mappings can be specified and changed at runtime without having to recompile settings. The mappings should be visualized and abstract from the programming (scripting) approach. Enough freedom should be given to extend the GUI to the needs of the current design tasks and to incorporate new ideas, concepts and devices.

Chapter 3

Related work

Different design concepts of user interfaces that support modeling and the representation of relationships between entities are analyzed. Another class of relevant graphical user interface concepts is also presented. Useful approaches that could be integrated into a graphical configuration support for the Patch Panel are summarized at the end. Although there are tools that deal with configuration issues in order to configure ubiquitous environments, none of them completely suits the needs of a Patch Panel GUI.

Survey in GUI concepts

3.1 GUIs for physical prototyping

There is a number of graphical user interfaces that support the physical prototyping process, partially by using plugins. In distributed environments, however, they suffer from certain drawbacks. Good concept but also cons of the different approaches are pointed out.

Drawbacks in distributed environments

3.1.1 d.tools

The tools prototyping environment presented by Hartmann et al. [2005b] focuses on product and interaction designers who posses knowledge about fabrication, content

Micro controllers

creation and interaction design but do not necessarily have insight into engineering aspects of design like programming micro-controllers, for example.

Programming via state machines

Confronted with a certain task that needs to combine several sensors and controllers, the developer can now connect all the physical parts of her prototype to a controller board to a computer where the behavior of the components is now graphically defined with the help of the d.tools modeling application (cf. Hartmann et al. [2005a]).

States and transitions

The application provides the developer with iconic representations of the physically connected parts. One representation stands for one state the device is in. By dragging connections between states and modules, a complete state chart is graphically modeled. The actual configuration of the physical devices via the controllerboard is then performed with the backend of the d.tools program. The developer does not have to care about this. Figure 3.1.1 shows an experimental setup for a media player prototype.

Virtual representatives of devices

Another important issue of the d.tools design is the loose coupling between the physical components and their virtual counterparts, i.e. that also without a physical connection to the machine the d.tools software is running on, the components and their behavior can be specified. The configuration is applied as soon as they get connected. Also several instances of one class of devices can be used. The system provides mechanisms to distinguish between the different components. The finished virtual prototype can also be tested virtually, again also without a physical connection.

Not extensible for iStuff

This prototyping approach is very open to new designs, the major disadvantage, however, is that variations or modifications for the iStuff approach cannot be applied to the framework. Another disadvantage concerning a possible integration of iStuff components is that the d.tools approach is based completely on specifying state machine behavior. iStuff can do more than state machines only.

Disadvantages of the approach

Since many representations of physical devices are drawn representing different states and they are connected via an arbitrary number of lines standing for state transitions, the

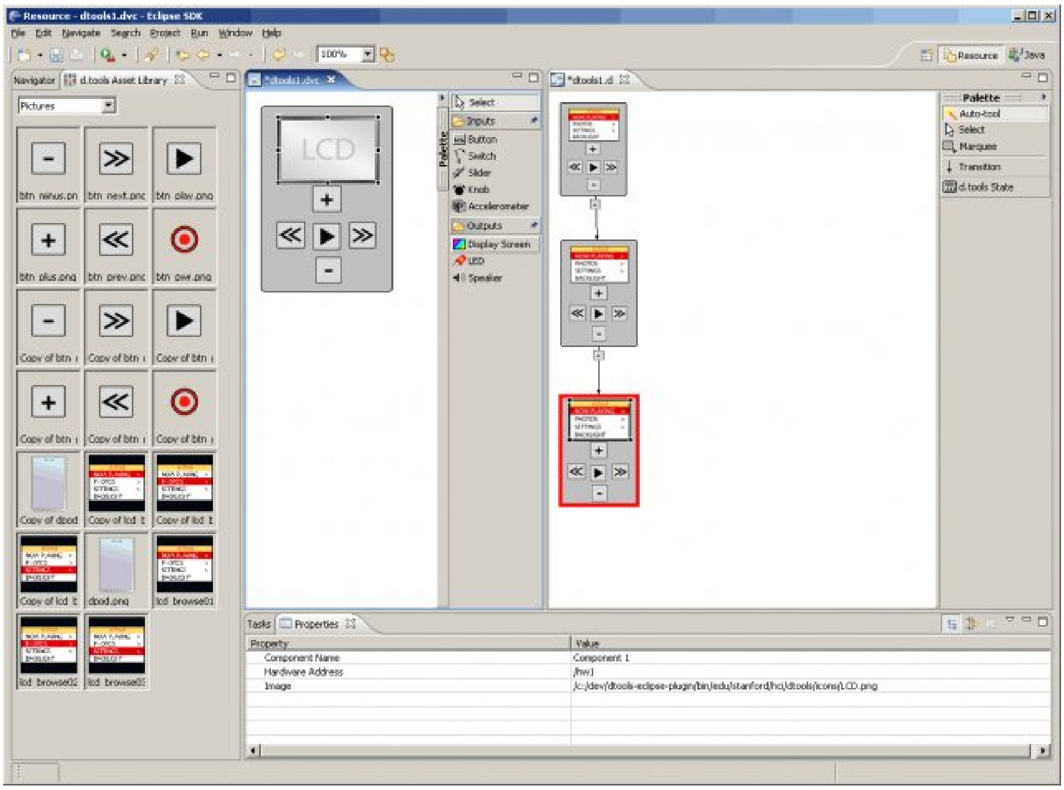


Figure 3.1: In the d.tools environment, state machines are defined with the help of graphical representations of real world devices. Connections represent state transitions triggered by one device.

general overview may suffer in large arrangements. The physical representation may also inhibit design iterations on the form factor because the component's appearance is fixed. It should be abstracted from the appearance and maybe the functionalities split up and modularized. As a last con, state explosion is to be mentioned. This may result in a big problem for interaction designers working without the help of developers.

Adaption of representatives

The idea of prototyping and testing without physical connections as well as the concept of representing functionality of real world entities with virtual counterparts should be adapted.

3.1.2 Max/MSP / pd

Data-flow metaphor

A very influencing concept of constructing and visualizing data flows is presented inside the application Max/MSP[1] and the open source project pd[2] , Max/MSP's open source counterpart.

Recombination in realtime

Besides processing MIDI (Musical Instruments Digital Interface) data, additional packages like MSP also allow combining graphical and musical projects. The main idea behind the graphical user interface is that different nodes represent different atomic functionalities like input and signal processing entities as well as components that are responsible for aural or graphical output. Other toolkits such as Phidgets and Teleo also provide extensions for Max / MSP. Nodes can be linked by dragging lines from output to input fields where automatic type checking prevents illegal connections. For example, if there is a MIDI generator that outputs MIDI values like note, duration and volume, these numbers can only be linked to a number processing node. A similar situation is shown in figure 3.2

Path concept and realtime changeability

After creating connections, the user can conceptually follow the "path" of an input signal running through the composition. Manipulation of the arrangement and the composition at runtime outline attractive concepts to be also applied for a rapid prototyping GUI.

No support for ubicomp environments

However, this approach is only usable for local compositions whereas the approach presented in this thesis deals with a distributed system supporting ubiquitous environments.

3.1.3 ICon - Input Configurator

Input adaptability

Dragicevic and Fekete [2004] tried to find effective alternatives for input devices. With the help of the *ICon (Input Configurator)* application, that addresses main input adapt-

[1] http://www.cycling74.com/products/maxmsp
[2] http://crca.ucsd.edu/ msp/software.html

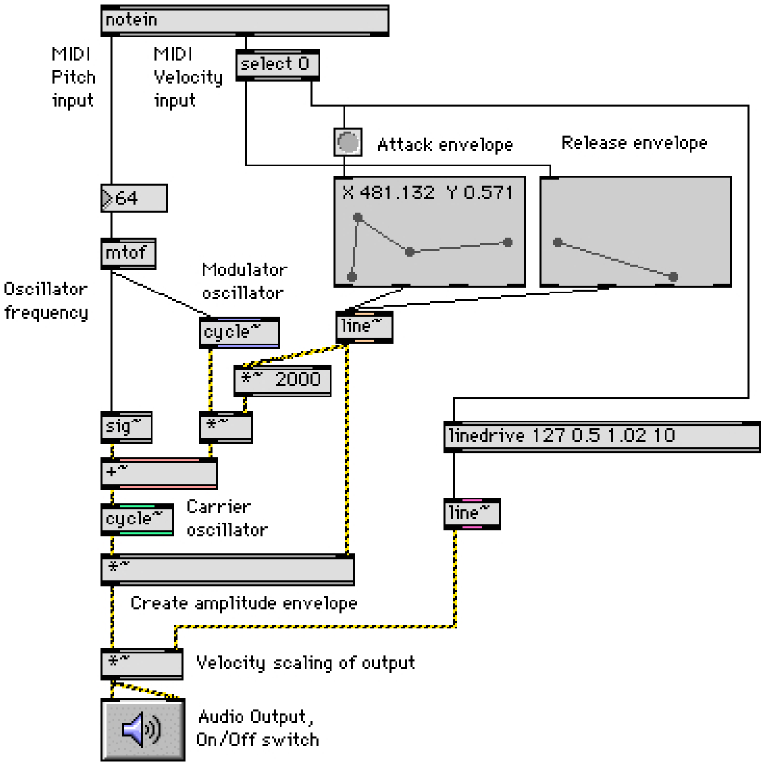


Figure 3.2: A sample patch in Max/MSP. The signal flow is top down, from left to right.

ability issues by making other applications fully input-configurable.

ICon devices

In order to specify new input behaviors, components can be arranged on a workspace. In the scope of the ICon approach, they are called "devices". ICon devices are abstract representatives of real world or input or output devices. Output can also be redirected to the system, e.g. in order to control the mouse cursor.

Classification of the components

The components are classified into three categories, depending on their purpose; some of them provide input data gained from a mouse or other external input device. Components that define internal transformations before the connection to an output device are also available. Output devices take parameters that they process and either redirect to a real world device or to a system resource.

Data-flow metaphor

This approach also makes use of a data-flow metaphor such that data coming from input devices can be directed to output devices and transformed on its way. Since each device provides or consumes different parameters, the parameters are separately available at input and output ports. Figure 3.3 shows an example of a data flow.

Interesting concepts for the Patch Panel GUI

The whole concept strongly reminds of Apple's Quartz Composer application (cf. section 3.3.3) but addresses a completely different application field. The concepts that are valuable for the Patch Panel GUI are those of providing possible data and parameters in forms of input and output ports and abstract representation of existing entities. The data-flow metaphor holds many benefits concerning the understanding of a composition and the separation of different functionalities provides a basis for reusing and recombining existing components.

Missing liveness

Although this approach provides very usable GUI concepts, it unfortunately lacks of liveness since the setup has to be run in order to be applied. Quick changes in the setup are only showing effects after a compilation phase in which the new mappings are integrated. This lack of direct application of changes may hinder the rapid prototyping process and the motivation of small changes may suffer.

3.1.4 Adobe Flash

Prototyping with Adobe Flash

Although Adobe Flash[3] was primary developed to create smaller animations and videos based on vector graphics, its development has reached a degree that enables it to be used a prototyping utility.

Powerful scripting language

For software applications, Flash can be used to create flat prototypes that react on inputs and perform predefined actions. More sophisticated interactions can also be created since the introduction of the scripting language *Action Script* which is part of every distribution. Thus, a conceptual image of the future software application can be created. For certain situations, it can even be appropriate to imple-

[3] http://www.adobe.com/de/products/flash/flashpro/

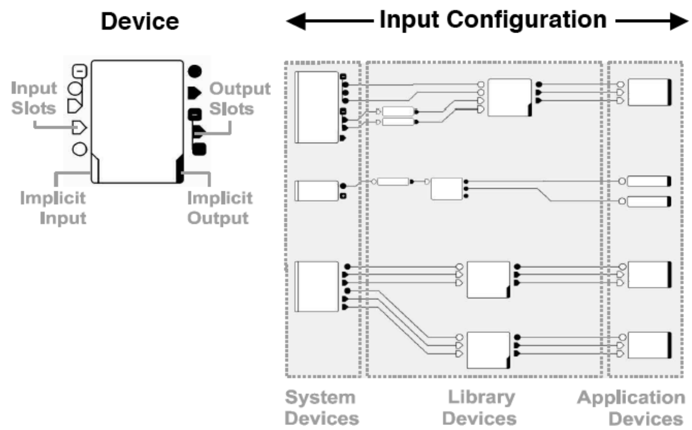


Figure 3.3: A screenshot of a composition with the Input Configurator (ICon) taken from Dragicevic and Fekete [2004]. Different devices are connected by taking values made available on outputs and linking them to inputs of other devices. In between, transformation can be specified with the same principle.

ment the complete application with Flash.

Plugins from hardware toolkits

More and more hardware toolkits like Phidgets, Teleo and the Calder-toolkit (cf. Lee et al. [2004]) provide plugins for Flash such that they can be used as input and output devices controlling Flash programs or receiving input from them.

Same application for different components

Flash is well known in the design community and by being extensible for new hard- and software components it is very flexible in terms of designing applications. The user, however, always works with the same application concept and does not have to learn another tool with each new toolkit.

Open and extensible approach

This open approach should be incorporated to the Patch Panel GUI such that easy extensibility is provided for any kind of prototyping toolkit that is integrated into the iStuff project.

Missing liveness and ubicomp support

Similar to the approaches presented in sections 3.1.1, 3.1.2 and 3.1.3, this tool as well does not really support prototyping in ubiquitous environments. Like the ICon approach new configurations can also not be applied at runtime but have to be recompiled in order to take effect.

3.2 GUIs for end users

Many restrictions

This class of GUIs makes much use of restrictions in order to guide users through their tasks. Of course, this concept also reduces the degrees of freedom a lot. However, this class needs to be paid attention to since the applications presented implement good approaches of simplifying the user interface and create levels of abstractions. Some concepts may certainly be useful for the Patch Panel GUI.

3.2.1 Jan Humble's jigsaw puzzle

Jigsaw puzzle metaphor

Humble et al. [2003] present the development of a user-oriented framework named *ACCORD (Administering Connected Co-Operative Residential Domains)* that allows easy reconfiguration of ubiquitous domestic environments. Lightweight components help to integrate a large number of devices that can be interconnected directly and are therefore configurable for different tasks. Examples were taken from security scenarios, where e.g. a surveillance camera takes pictures if a movement was detected and the recorded picture is sent to the house owner's mobile phone. Another example included a household scenario in which certain grocery items are ordered as the stock is depleted.

Specification behavior

The system is developed for end-users and thus emphasizes ease of use. As a consequence, the reconfiguration of integrated devices only allows a small degree of freedom. Scenarios are arranged with the help of an editor in which the connectable devices and services are presented as icons.

Combining puzzle piexes

The key problem, namely that not all connections cannot be meaningful, is solved by applying a jigsaw puzzle metaphor; the iconic representations differ in their shape and so it can be determined whether a component provides, transforms or consumes data.

For example, a motion sensor is naturally not able to process any input and only provides sensor data. The other way round, a device that provides a certain service like an ordering service does not provide any output that could be meaningfully connected to any other component. By using different shapes, the functionality provided by a component can easily be illustrated. Like pieces of a real jigsaw puzzle, the icons either have a flat side on the left and a nose on the right, vice versa or a whole on the left and a nose on the right side. These shapes stand for data providing entities, consumers that handle incoming data or data transformers, respectively.

Shapes clarify semantics

Figure 3.4 shows the described basic shapes and an scenario where the door bell does not provide a whole for connecting other pieces to it, but only a nose that indicates a hook for connections with different components providing a fitting left side. To that hook, a photo camera is connected as an intermediary device that takes data and passes it to the next chain member, a PDA that receives the camera data and displays a photo of the visitor.

Door bell scenario

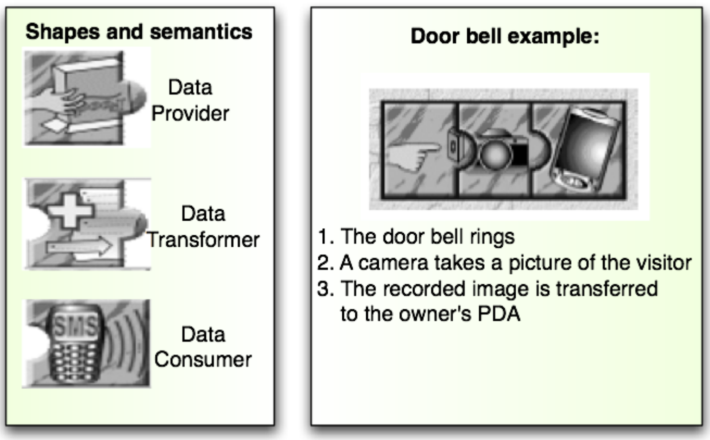


Figure 3.4: On the left: The different kinds of components with their according shapes. On the right: The security described scenario as the user sets it up in the editor.

With that mechanism, new scenarios can be created quickly and easily by combining pre-defined puzzle pieces. As user studies prove (cf. Humble et al. [2004]), the design concept

Not applicable to custom GUI

is widely accepted and liked, the metaphor is useful. The disadvantage of the jigsaw solution is its limitations for the developer who needs higher degrees of freedom this system does not provide as it is designed for the end-user who should only apply pre-defined functionality. Real Prototyping work is not possible as internal design decisions cannot be made.

Ambiguity of interpretations

Another problem rising from the simplicity of the concept is, that only one interpretation of the meaning of "connect" is allowed. This can be ambiguous sometimes as in the example depicted in figure 3.4. This scenario could be interpreted in different ways. For example, one could think that the doorbell activates a trigger that lets the photo camera take a picture of a PDA instead of sending its data to it. The applied metaphor is very user-friendly but ambiguities should be eliminated in order to provide easy to interpret compositions.

3.2.2 CAMP - magnetic poetry

Magnetic poetry metaphor

With the *CAMP (Capture and Access Magnetic Poetry)* research project, Truong et al. [2004] tried to build an end-user application that allowed the easy configuration of ubiquitous environments. A new way that enables the users to achieve their design goals in terms of specifying them instead of forcing them to think about detailed devices configurations and combinations should be found.

Fixed vocabulary for building blocks

The CAMP user interface was a step into that direction: A fixed vocabulary presented as magnetic poetry building blocks is presented to the user from which she forms sentences that describe the desired behavior of the system. User only describe their aim and do not have to care about the devices and connections involved. This is the task of the underlying system. Because of the fixed vocabulary and a quite tight design space, a first prototype could be realized.

GUI based on the INCA system

With a GUI based on the INCA system (cf. Truong and Abowd [2004]), users build their task definition out of presented building blocks just like with real magnetic poetry pieces. The browsing of available blocks is supported by categorization and color coding. The set sentence is then interpreted by the system, redundancies and conflicts are resolved based on certain assumptions. After the processing the system generated sentence based on the building blocks is presented to the user, then the devices needed for the task and their configurations are setup automatically. Figure 3.5 shows the user interface.

Narrowed design space

For the prototype presented in the UbiComp 2004 paper, the design space was narrowed to the field of video capturing scenarios in order not to make the system not too complex and retrieve first results from an easier to implement prototype.

Alternatives for presenting the vocabulary

From the results of that work, modifications that include different metaphors to specify the design goals like comic strips should be implemented in order to learn more about the effects of different presentation methods of the working vocabulary.

Limitation to fixed vocabulary

Interesting about this work concerning the planned GUI is to present a vocabulary to the user that is manageable in terms of size. So, the learning curve can be kept lower because the user only has to work with a fixed set of functionalities that can be recombined and even be abstracted.

Set of atomic funcitonalities

Providing a set of atomic functionalities seems to be a very useful way to keep systems flexible and extensible. For the goals of this thesis, the end-user friendliness is not well suited because the iStuff project focuses on designers that need to specify tasks and behavior of ubiquitous environments. A system that automatically configures all devices involved would narrow down the design flexibility and leave out the design task.

3.2.3 iCAP

No need for writing code

The *Interactive Context Aware Prototyper* (iCAP) allows prototyping for context aware-applications Sohn and Dey [2003] tailored to the end-user by avoiding the need of writing code. A ubiquitous application is specified by creating rules based on IF-THEN clauses, relationships-based

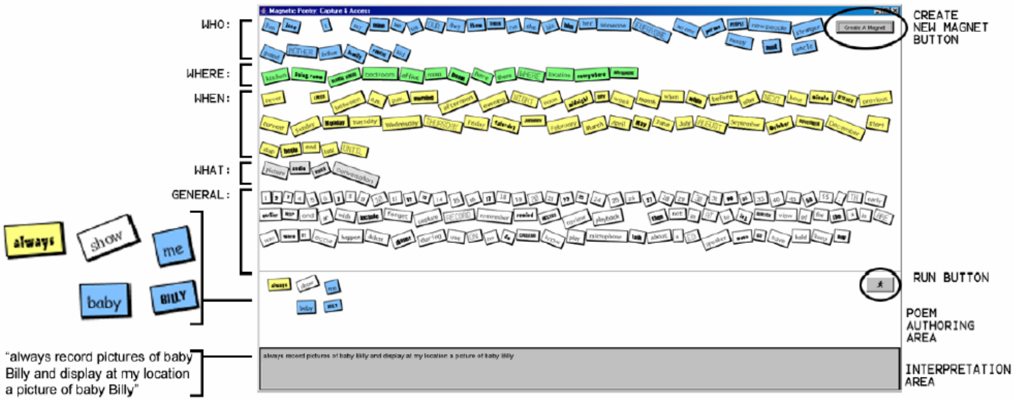


Figure 3.5: A quite simple GUI lets the user specify sentences from a fixed vocabulary represented by little building blocks that can freely be combined Truong et al. [2004].

actions and environment personalization. This rule-based system allows a higher degree of flexibility but also introduces more complexity. The whole prototyping process is visualized and users choose from sets of predefined components. This makes specification a lot easier. Figure 3.6 shows the iCAP interface.

Visual representation of rules

Again, the idea of narrowing down the degrees of freedom in terms of the choice of components is applied. The enrichment of specifying rules appears as good help to prototype more complex scenarios. Although it seems to be more difficult to parse the scenario. From the iCAP work, the concept of introducing rules and restricting the choice to a fixed set of possibilities was found to be useful for the custom work.

3.3 Other relevant GUI concepts

Interesting GUI designs

This section presents ideas and concepts from different research fields not directly connected to prototyping in ubicomp environments. Since they present improvements concerning the presentation of information graphically, they should also be examined in order to extract some general GUI design concepts.

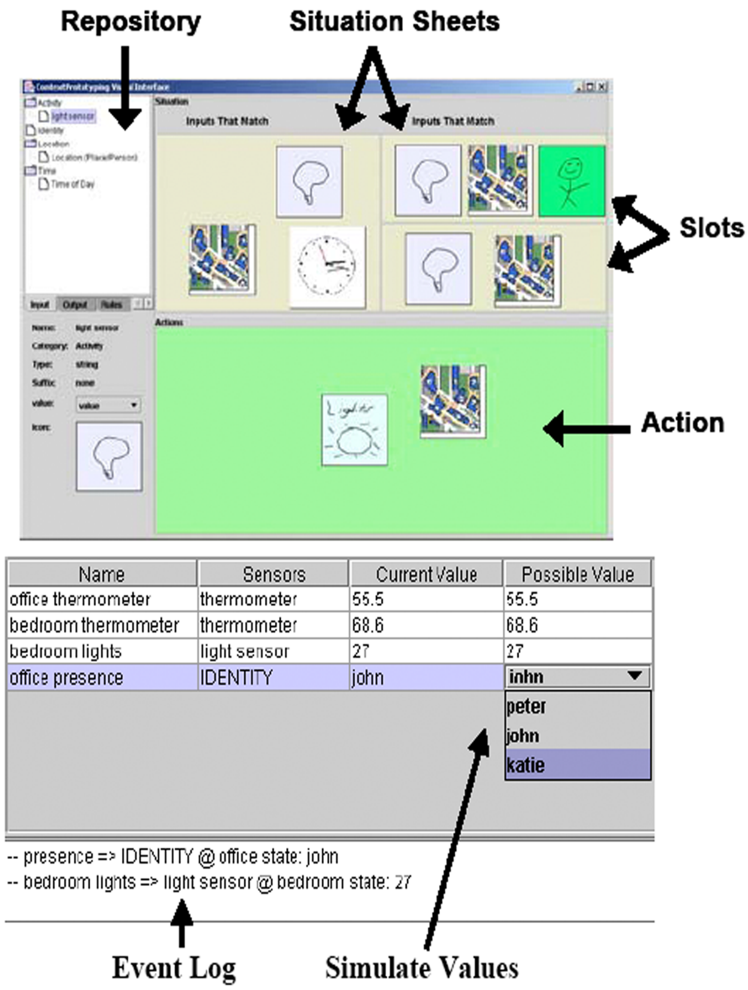


Figure 3.6: The iCap drag&drop user interface (cf. Sohn and Dey [2003]). The lower part shows the rule editor.

3.3.1 XML schema mapping visualization

Visualization of XML schema browsing strategies

Robertson et al. [2005] analyze ways to improve the overview of XML schema mappings as, with increasing size of XML schemas, the visualization of a mapping is often hard or even impossible to parse for the reader. Figure 3.7 shows an example of the visualization of a large mapping. In order to solve that problem, the authors implemented new functionalities into an existing XML schema browser.

Highlighting and automatic scrolling

The selected element and its counterpart in the compared schema are *highlighted* and *centered* on the screen by scrolling automatically to their position. Thus, selected elements are always found in the vertical middle of the screen and can directly be spotted as they are marked.

Tick marks and coalescing trees

Additionally, search hits in the whole schema are presented as marks on the scrollbar at the window side. These *tick marks* are color-coded in such a way that already selected hits appear in a different color from those currently not selected. Momentarily irrelevant information is hidden by making use of *coalescing trees* and therefore becomes more compressed.

Selecting and searching

Features like *multiple selection* as well as *incremental search* support the schema browsing process.

Bending links avoid misinterpretations

Another very important issue concerning the arrangement of nodes and links is the idea of *bending links*. With this feature, visual ambiguities can be resolved as sometimes a link lies behind another node that makes it impossible to decide whether the link belongs to that node or if it is only covered. Bending a link that is covered by a node presents a reliable solution to determined how the connections is to be interpreted.

Visualization aids interesting for the custom GUI

This work presents a lot of visualization aids that were accepted by users (proven by user studies in the article cited). Almost all of these approaches seem to be well suited for the planned GUI, especially the idea of hiding momentarily irrelevant information and bent links in order to avoid occlusion.

3.3.2 Photo Mesa - zoomable image browsing

Effective approach

In their article "Does Zooming Improve Image Browsing", Combs and Bederson [1999] describe an image browsing system, in which a large collection of images is presented in a thumbnail-like view.

Panning and zooming

Instead of selecting a thumbnail from a list, like in conventional image browsing applications, users can navigate

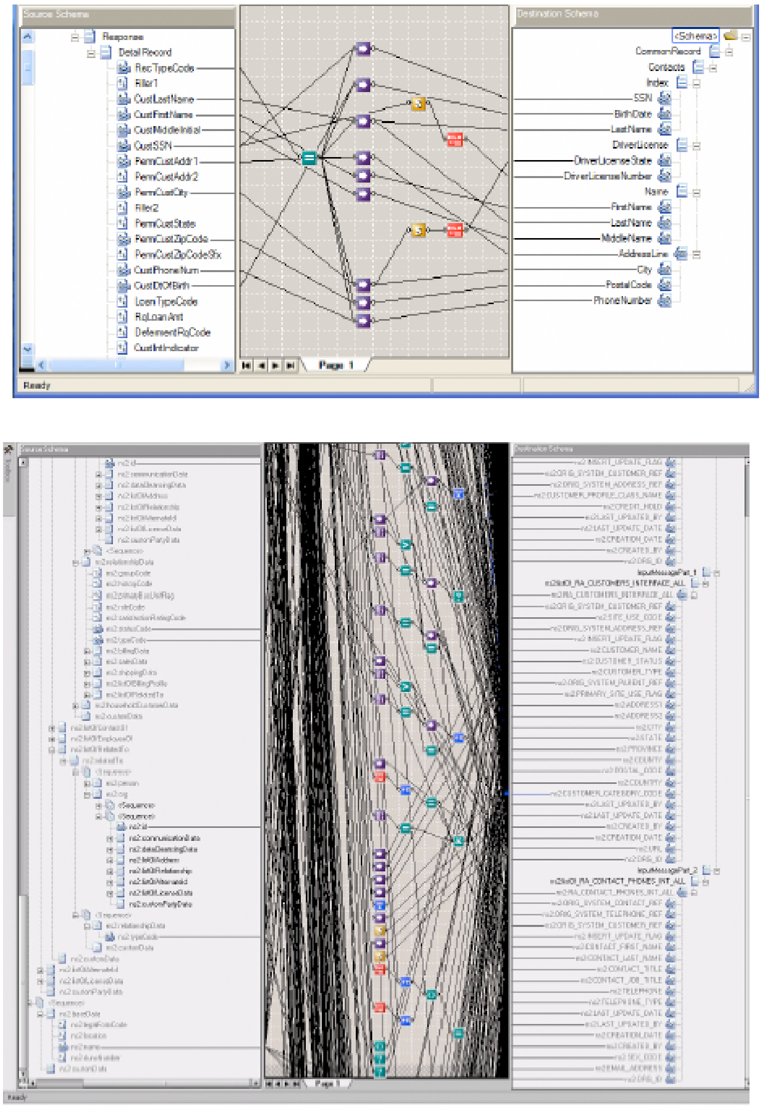


Figure 3.7: Upper part: A small XML schema mapping where the old visualization technique is applied. Lower Part: The same technique does not scale with large mappings.

through the available images by panning across and zooming into them. This approach utilizes the human capability of spacial memory. That means that the user keeps the orientation of the collection and knows roughly about the position of the other images. She can rely on her spatial memory to quickly retrieve images she has seen before in-

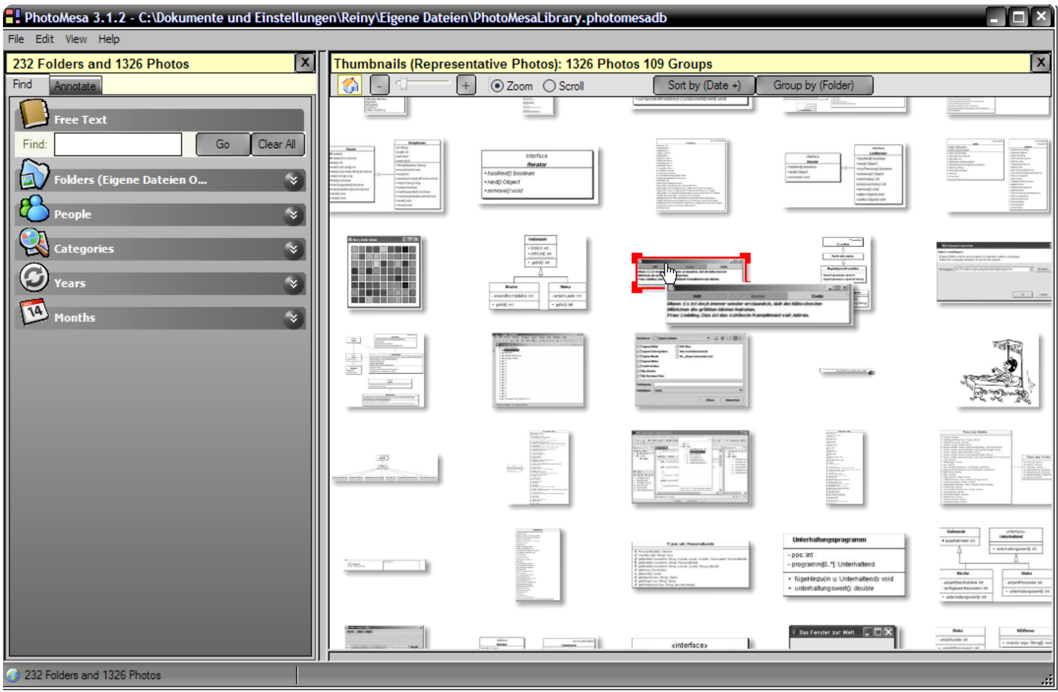


Figure 3.8: The UI of the Zoomable Image Browser Photo Mesa allows panning and zooming into the thumbnails.

stead of searching through a list.

Comparing by zooming

The task of browsing through an image list is also supported if the user roughly knows what she is looking for. By presenting a collection of images in a zoomable thumbnail-like view, different images can directly be compared. If certain details of an image need to be displayed in more detail, one can simply zoom into the view up to certain degree without loosing eye contact to all of the other images. Whereas traditional image browser only allow fixed degrees of enlargement (mostly thumbnail and nearly fullscreen), the zoomable image browser allows any enlargement in between.

Concept scales up to 225 pictures

User studies revealed that this concept is applicable for up to 225 pictures. Figure 3.8 shows a screenshot of the zoomable image browser application “Photo Mesa” that is part of the research work. Further development of this browsing method was encouraged by the experimental results.

The idea of zooming into relevant information seems to be applicable for a Patch Panel GUI - at least less relevant information should be hidden in different perspectives. A rougher "zooming" like the one presented with macro patches in the Apples Quartz Composer (see section 3.3.3) is similar to the presented approach although not as smooth as real zooming. Such a way of supporting the user who has to process the information presented, however, could be useful.

Zooming may become relevant for large compositions

3.3.3 Apple Quartz Composer

The Apple Quartz Composer[4] is part of the Developer Tools since Mac OS X Tiger (10.4). This application is intended to provide an easy way to create screen savers or graphical animations that are controlled in realtime. "Patches" are arranged and combined on a workbench. They provide different functionalities and can therefore be differentiated between *generators*, *modifiers* and *outputs*.

Complete abstraction from programming

Generators provide values from system devices like mouse or keyboard. MIDI values can also be caught from the built-in ports as well as audio signals recorded by the system's sound device. That means generators provide values that can be transformed by modifiers (through calculation, type conversion, logical formulas, etc.). Therefore, generators usually only possess output ports, where values can be taken from, whereas modifiers often have both: input and output ports. Patches that provide the interface to other system components like the graphics engine (which is the intention of the program) only have input ports. Figure 3.10 shows the Quartz Composer editor and the viewer window where graphical output is shown.

Different types of functionalities

The values they receive are processed internally. One simple example is the connection of a generator that provides mouse coordinates. These values are directly connected to the x-position parameter of a patch that renders the application's logo into a viewer window. As the mouse is moved, the image inside the drawing window also moves depend-

Example with mouse coordinates

[4] http://www.developer.apple.com/quartzcomposer

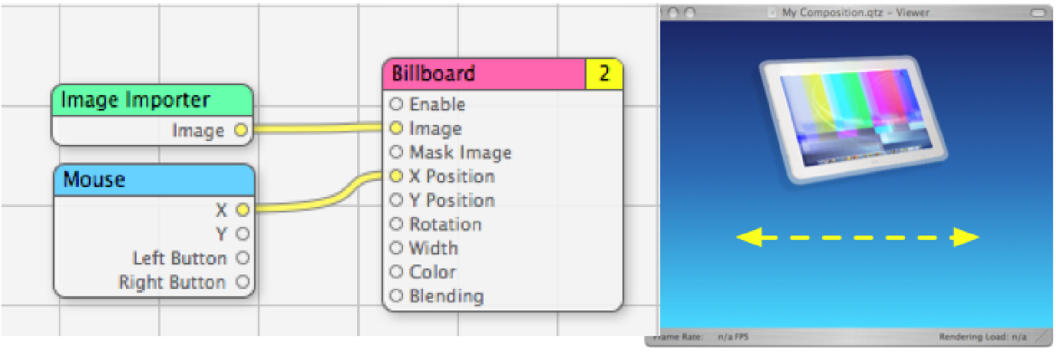


Figure 3.9: One example of a small Quartz Composer Patch. Mouse and image data are provided by the left two patches. The right "Billboard" - patch takes image and mouse position data and passes them to the operating system's graphics engine. That way, the image is moved together with the mouse.

ing on the mouse movement and eventual calculations performed in between. The "Billboard' 'patch internally calls methods from the operating system's graphics engine (cf. Figure 3.9).

Key features

Tutorials on Quartz Composer[5] provide more detailed information about the use of the application but the most interesting key features should be pointed out here:

Drawing connections

- *Drag&drop*: clicking on one output port and dragging the mouse causes a connection to appear that can be dropped onto another input parameter. Usually, patches do not allow connections to themselves.

Type checking and conversion

- *Implicit type-checking*: If the data types of a connection fit, the connection line is drawn in yellow, if the data type do not exactly match, but the connection can be made (e.g. integer outputs are connected to boolean inputs), the line is drawn in orange. This directly indicates the type mismatch. If data type absolutely cannot be combined with each other, e.g., strings are connected to integer values, the connection stays white and disappears as soon as the mouse button is released.

Macro patches

- *Abstraction*: A composition can be abstracted by mak-

[5] http://developer.apple

ing a *macro patch*. The single patches and collections are collapsed into one patch that is displayed from now on. A hierarchy browser helps to keep the overview. This concepts allows to have compositions inside compositions. Ports that are needed can be published such that also a connection to the environment beyond the patch scope can be established.

- *Incremental search*: Browsing through patches is facilitated by an incremental search engine where users can enter parts of the patch's name or description into a search field. With continuous typing, the search results get narrowed.

Incremental browsing

Quartz Composer includes a lot of the features regarded as useful in the overview of related work. After presenting paper prototypes that were created in a first design iteration for a Patch Panel GUI, the desired functionality for the planned GUI and the features provided by Quartz Composer are compared again. The results of that comparison are presented as a second result of a design iteration in chapter 4.

Many extracted features are provided

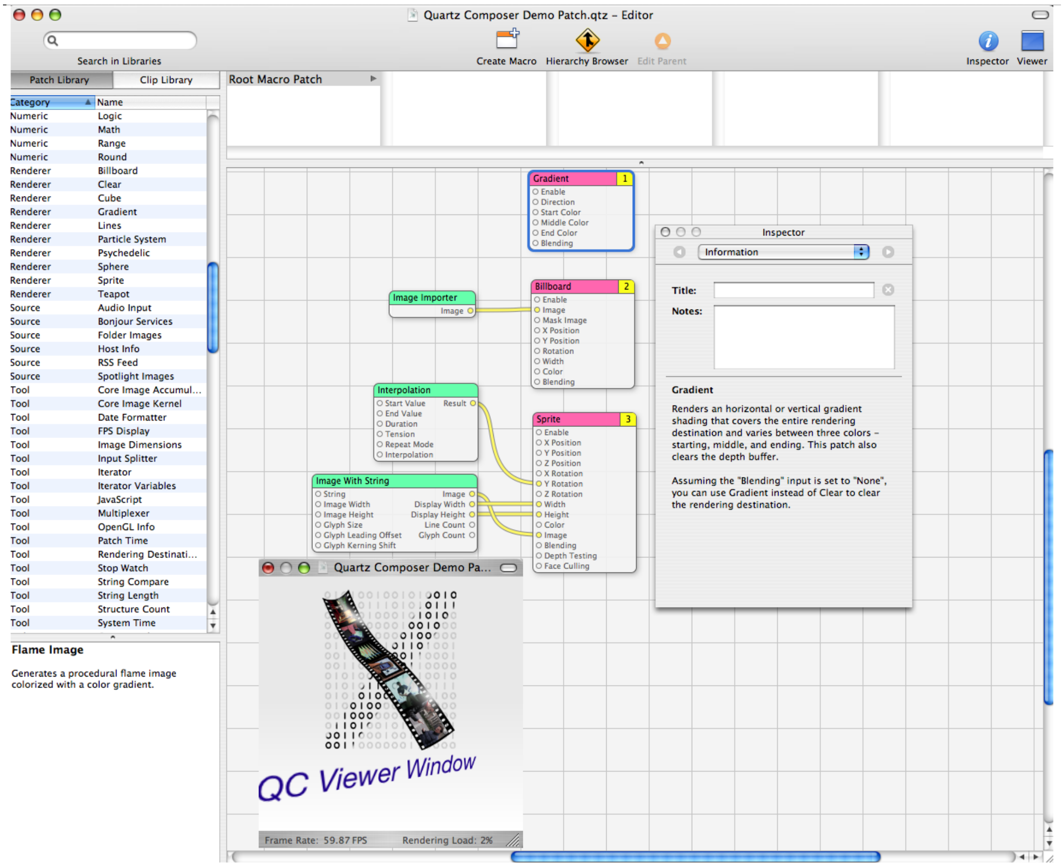


Figure 3.10: The Quartz Composer workbench. On the left, a palette holding all available patches is shown. The text field at the top of that list allows incremental searching. Below the list a description of the patch is displayed. The upper part in the middle of the screenshot shows the hierarchy browser which only shows the root patch as no macro patches have been created yet. Below there is the work area with a basic patch that creates a rotating string below the logo. This output can be seen in the lower part of the screenshot in a floating window. On the right, the Inspector window is placed.

Chapter 4

Collecting Concepts: Patch Panel GUI prototypes

In the preceding chapter, different GUI concepts supporting prototyping processes or dealing with visualization issues, respectively, were discussed. This chapter summarizes those ideas that seem to be useful for a graphical user interface for the Patch Panel. Custom ideas are also integrated into the summary. After presenting a collection of concepts, a GUI prototype that resulted from the initial design phase is presented together with the results of its evaluation at the end of this chapter.

Features and storyboards for the planned GUI

4.1 Ideas and concepts

This section discusses features found in related work as well as custom ideas in more detail. The relation to a possible integration into a Patch Panel GUI is always drawn.

Feature from literature and custom ideas

4.1.1 Preliminary design patterns

In form of a feature list, different ideas are described in

Rough design patterns

more detail and compared with the planned Patch Panel GUI. The feature list should be regarded as a rough collection of design patterns as they can be found for software construction in Gamma et al. [1995] or for interaction and website design (cf. Borchers [2001] and VanDuyne et al. [2002]). At this stage the collection is kept as a list and not structured to represent a complete pattern language but proven concepts are collected and could be summarized for future purposes concerning similar design tasks.

Composition

Reusable components

Users connect entities by dragging lines whereas each entity provides a special functionality. This encapsulation allows an arbitrary number of recombinations in such a way that new tasks can be solved by reusing components. For the Patch Panel, entities of different iStuff components are needed that wait for specific events from the Event Heap, other entities that allow transformation of the information received and yet another class of entities that post events to the Event Heap. The software proxies connect to the Event Heap and subscribe to certain event types or post events to it.

Easy retrieval of components

Fast search

The user should be supported when browsing through the available components in form of a tree view or any other kind of list that categorizes them. Expandable trees, column browsers like in the Apple Finder or incremental search as it is provided by Quartz Composer seem to address this issue in a promising way.

Drag&drop support

Entity arrangement

A list of available entities is presented to the user in almost all GUIs. There are several ways the selection process is implemented. Max/MSP allows the user to drag and drop a basic entity. Then, it is specified by typing its name. iCap or Quartz Composer provide incremental search to narrow down the search space. From the displayed list, objects should be added to the composition by double-clicking or dragging them into the workspace.

Avoidance of illegal connections

Connection checking

In the jigsaw puzzle approach, the group around Jan Humble gave natural hints of the possible ways to connect different components. These hints came in the shape of noses and holes like in real jigsaw puzzles. With the help of these hints, users knew directly what pieces would fit together and how they are to be arranged.

Different restrictions

Applying the jigsaw metaphor, the task of connecting and arranging becomes familar. Other GUIs of that kind like Max/MSP or Quartz Composer make use of similar restrictions but it is not always clear at once, why a certain connection is invalid. It is only shown that something is wrong. Faulty connections are rejected by not being drawn. First a manual check of the data types that should be connected reveals the reason for the rejection. For the Patch Panel GUI, implicit type checking is necessary otherwise there was no improvement compared to the type checking mechanism integrated into the original Patch Panel scripting language.

Automatic type conversion

Feedback on errors

While connections are drawn, the types that the users wants to connect should directly be checked. Like mentioned above, some GUIs do not accept connections if they are not valid in the sense of a type mismatch. An important issue is to provide feedback for the user why a connection could not be made.

Color coding

Quartz Composer makes use of a color coding; all connections are drawn while the user drags the mouse and holds down the mouse button. If a connection could be established, valid ones are directly marked as yellow and drawn as soon the mouse button is released. If there is a type mismatch that can be resolved, the line is marked as orange. An example for such a type mismatch is the connection from an integer output to a boolean input. Although boolean inputs expects the values 0 or 1, integers that are greater than 0 are interpreted as TRUE and 0 is interpreted as FALSE.

Implicit conversion

This implicit conversion makes it easier for the user to connect even non-matching types without having to manually care about such simple and often occuring type conversions. Connections that cannot be converted remain as white and disappear after the mouse button is released. An example for this are inputs that expect integers. Connections coming from string outputs cannot be connected. The opposite direction, however, is possible since integers can be converted to strings.

Consistent flow of information

Data-flow metaphor

In Max/MSP, Quartz Composer and other GUIs, connections can only be drawn into one direction. That means that connections start at outputs from entities and end in inputs of others. This consistent flow of information is very important because the user is enabled to build up a mental model of the composition's functionality. With this metaphor, it is easier to see what kinds of transformations are applied to the data and to estimate the results.

Liveness of changes

Changes at runtime

For the Patch Panel GUI, there should be no need of "compiling" the composition. Changes made should directly be available at runtime such that the resulting effects can directly be perceived. This is a very important feature for rapid experimentation and encourages more design iterations and the exploration of the effects also caused by little changes in the setup. iCap, d.tools, Max/MSP, pd and Quartz Composer provide direct incorporation of changes in the setup. The original term "liveness" was created by the *Morphic toolkit* described by Maloney and Smith [1995].

Abstract representation of real world entities

Representatives

In the d.tools approach, entities that can be connected are abstract representations of real world objects. With the help of these abstractions, the user does not have to care about internal functionality. The abstractions delivered are complete packages that describe the functionality the objects offer and the kind of data they can process. Even if the real world counterparts are not available at composition time, the abstractions can be used in the design process. In the planned GUI virtual setups that do not require all components connected to the system at runtime should be supported.

Provision of template values

Specification of templates

There are situations where not all input ports need to be connected to another entity but fixed values for these ports are needed for the functionality of the entity. Max/MSP and Quartz Composer provide that mechanism where input ports that are not connected can be set manually. In order to play a MIDI note, for example, other value besides the note value are pan and duration. These could be set to an invariant value in order to play all notes with an equal duration and pan location. This concept could be applied to the Patch Panel GUI in order to set specific event fields or to compare incoming data with fixed thresholds, for example.

Abstract testing

Virtual testing

In d.tools, it is possible to test the composition even if the real objects that are represented are not connected or just partially connected. A debugging mode makes it possible to follow the state transitions after a certain user input graphically. With that concept, the prototype can also be tested virtually. This feature is a direct addition to the abstract representation and should also be considered for the project since it essentially supports the rapid prototyping process.

Highlighting current selections

Marking connections

In the XML schema approach, currently selected connections where graphically highlighted whereas connections that were not belonging to the mapping were greyed out. That makes it easy for the user to keep the overview if there is a large number of connections and entities to parse. For the Patch Panel, that would mean that connected entities are highlighted whenever another component connected to it is selected. As a possible result from that technique, the overview might suffer since too many entities were highlighted. Maybe it would be better to highlight connections only when users clicks on them. This had to be examined in further evaluations.

Occlusion avoidance

link bending

In the same approach as in the last paragraph, occlusion of connections made it impossible to distinguish whether a connection belongs to the occluding entity or if it just runs "behind" it. Link bending was the strategy applied to that problem in the XML schema visualization approach. It would be a useful feature for the planned GUI and other ones, too, although it is not implemented in many tools.

Panning and zooming

3D-browsing

The three-dimensional image browser "Photo Mesa", presented in section 3.3.2 provided panning and zooming. This approach could also be useful in order to gain overview of the general concept of a composition. Details are revealed by zooming into the composition and a specific component. It is to be found out at what composition size the feature really becomes useful.

Abstraction

Hierarchical ordering

Another concept that might fit well into the scope of the planned Patch Panel GUI is the possibility of hierarchically ordering clusters of components. Quartz Composer, for example, allows this by giving the chance of packing entities and their connections together in one large patch, called a *macro patch*. If a connection to another hierarchy level is needed, inputs and outputs can be published and so be accessed from other levels of abstraction. A hierarchical browser allows drilling down into the patches from the root pane. Macro patches themselves can also contain other macro patches such that the abstraction is unlimited.

4.1.2 Adding custom ideas

Integration of custom concepts

Starting with a survey on existing GUI concepts, own ideas also raised while trying to apply interesting concepts to the Patch Panel GUI. They shall be named as keywords in this section. Most of the ideas are directly related to the iStuff project such that they cannot be applied to GUI concepts in general.

Overview window

Map of composition

A window that provides an overview of the complete setup can help navigating through large compositions. As soon as the user hovers over the floating window the mouse cursor turns into a hand symbol that allows - similar to the Adobe Reader[1] software to pan the actual view by clicking and dragging it. This concept is also often used in geographic applications and navigation services like Map24[2]
.

[1] http://www.adobe.com/products/acrobat/readermain.html
[2] http://map24.com

List currently running proxies

States of proxies

A window that holds information about currently running proxies and their Event Heap connection should be provided. Otherwise it might become hard for the user to figure out what components are active and ready to receive or post events.

Status of currently used entities

Entities in a composition

It would also be helpful to know what iStuff abstractions are currently being used and whether they are connected to the Event Heap. Internal information like a component's ID should be accessible inside the status view. Also the connection status should be cared about in terms of selecting from available Event Heaps and managing the component's connection. These settings should be applicable to single or all entities.

Iconic representations and custom names

Renaming and provision of icons

The used components of a setup should be represented as icons, such that from within the whole composition is easy to parse what kinds of components are used. There should also be possibilities to rename entities and add additional information to them. Buttons, for example, could be named by their color and not their numerical order after which they were added to the compositions. This feature improves the readability of a composition since component names would indicate their purpose by their name, similar to well-named variables in programming languages.

Generation of events and values

Event factory for debugging

In order to test configurations, the generation of specific events and values may become useful to see the data processing inside a Patch Panel mapping. The iROS package already provides such a mechanism as a command line utility. Maybe a wrapper around that application or methods to directly feed numeric, boolean or string values into components arranged inside the Patch Panel GUI will support testing compositions.

Graphical visualization of values

Plotting values

A graphical visualization in form of charts or a plotter could be helpful when tasks occur in which thresholds need to be determined or the progress of input and output data, respectively, has to be monitored. This may be the case when sensor data is collected.

4.2 First prototypes

Patch Panel GUI prototype

After a first DIA-iteration that collected existing prototyping concepts, a paper prototype was developed and evaluated in a discussion with other application designers at the department. The prototypes came in form of storyboards that showed the interaction flow by means of specific scenarios motivated by earlier work performed by Ballagas et al. [2003]. Some examples of the originally developed storyboards that helped to create a conceptual picture of the planned GUI and its functionality can be found in appendix A.

4.2.1 General Patch Panel GUI concept

General layout

The general layout of the paper prototype GUI is shown in figure 4.1. On the left, a tabbed pane lets the user choose between iStuff hardware or services, respectively, or "mediators", that are responsible for type conversions and mathematical transformations.

Available components

The buttons below divide the iStuff components into devices that are available in the ubiquitous environment and machines that provide services to control applications running on them. These machines may also act as proxies for lightweight components like sensors as discussed in chapter 2.

Tree view

Depending on the selected tab, a tree view is shown that arranges devices and services according to their data flow direction (input / output), name and class (iButtons, e.g., belong to the general class of buttons).

Smarticon pane

The top row is reserved for a smarticon bar where often used commands can directly be accessed via iconic representations, the lower part of the application window is reserved for messages that give hints on solutions of possible conflicts, similar to the Eclipse IDE[3] .

Workspace

Most of the application window space is consumed by the workspace, where compositions are created and arranged. On the right hand side, a collapsed floating window is drawn that provides an overview of the whole patch. A smaller rectangle reflects the current position on the complete workbench.

4.2.2 Interaction illustration

Storyboards

To illustrate the interaction with this prototype version of the Patch Panel GUI, two examples of iStuff scenarios are presented and virtually implemented inside different storyboards. Only relevant steps are shown in order to provide a feel for the application behavior. Many more interaction concepts were worked out during the storyboard development process that also referred to the feature list presented in this chapter but because the storyboards were not developed further and implemented, only a small excerpt is shown. Further examples in form of the original hand-drawn storyboards can be found in appendix A. The reasons for the abortion of the development process for a new Patch Panel GUI is the Quartz Composer application that not only looks quite similar to what was planned but also provides most of the features desired (see chapter 5).

[3] http://eclipse.org

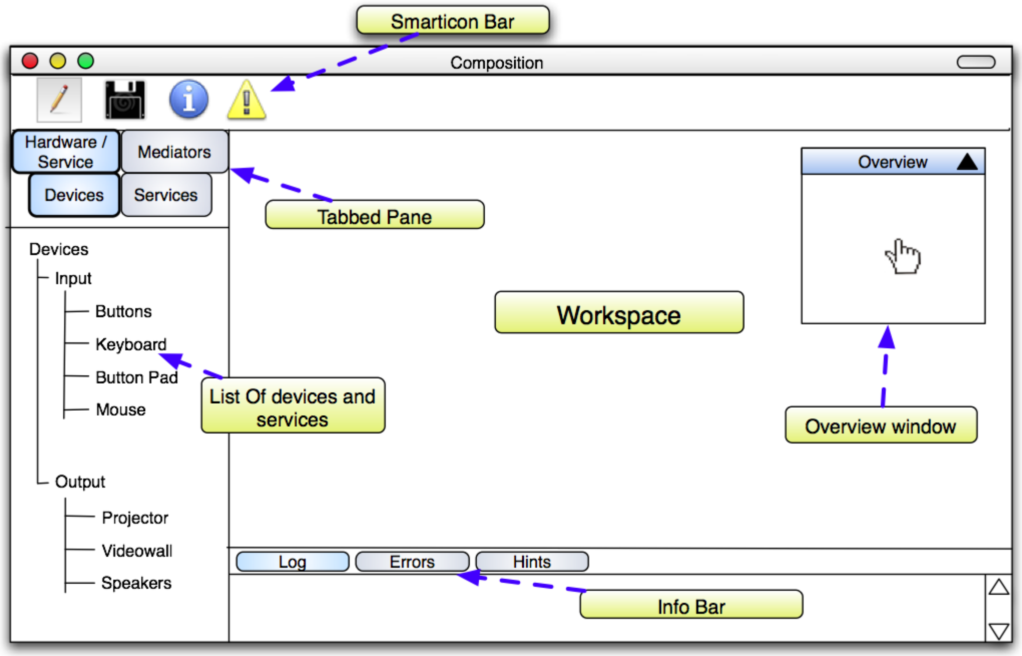


Figure 4.1: The storyboards prototype for the Patch Panel GUI showing all the parts described.

Toggle button example

An iStuff button is to be configured to act as a toggle button that controls the room lights. In the conventional scripting approach, the developer would have to manually implement a state machine, that changes its state with every button press that is sent (cf. scripting language example in chapter 2.2.3). Depending on the state it can be determined what kind of events have to be sent by the Patch Panel.

States for a standard button

In this example, an iStuff button should turn on and off the ceiling lights. Figure 4.3 shows the storyboard that illustrates the undertaken steps to create that setup. The original storyboard is shown in figure A.1 which has been recreated for better readability. The final composition is depicted in figure 4.4.

Toggle button example

1. A red iButton and ceiling lights were selected from the tree view on the left and placed onto the

Selection of components

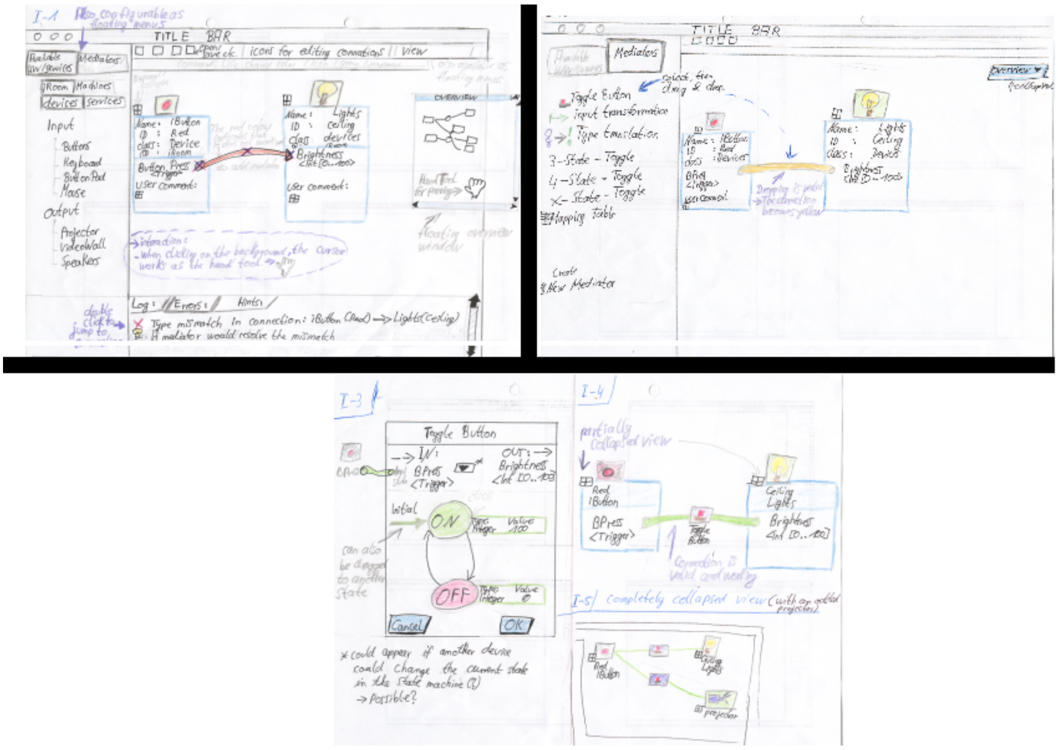


Figure 4.2: The original storyboard for the scenario described in section 4.2.2. For better readability this storyboard has been recreated in figure 4.3.

workspace. The available outputs and inputs with their corresponding values are illustrated as circles. The iButton is capable of sending triggers, the lights controller representative processes integers in the range from 0 to 100. Each components provides collapsed views that provide additional information or let the user edit component-specific settings. A small icon in the upper left corner provides quick information about the kind of the component.

Type mismatch

2. The user tries to connect the outlet of the iButton with the input port of the lights. Since the variable types to not match, the faulty connection is indicated in red color and a message is show in the log bar below.

Mediator

3. The user's next step is to select a mediator from another tree view in the tabbed pane and drag and drop it onto the erroneous connection. The Patch Panel GUI generates a state machine with two different states as shown in the next step.

4. A window with a simple state chart editor opens. The user specifies the initial state by dragging the arrow labeled "Inital" to the desired one. As the inputs can only be of type "Trigger" the state transitions do not have to be labeled. At the upper part of the window the incoming and outgoing connections are shown.

State chart editor

5. The whole view of the configuration is shown where the state machine is build into the connection which now appears as completely valid, indicated by the green color (cf. figure 4.4).

Final setup

Controlling a music player application

In this example, a new way to control music playback is to be found. A software proxy that controls a music player application is controlled with sensor data provided by a Phidgets Interface Kit. This kind of sensor board provides eight input ports that receive analog data provided by other Phidget sensors like a touch, force or rotation sensor as well as an analog slider. With this setup, only one software proxy posting events of type "Interface Kit" is needed. Figure 4.5 shows the sensors that should be used. The storyboard showing the configuration process of this scenario is depicted in figure 4.6, because of readability issues only four output ports are drawn. The final set up of this scenario can be seen in figure 4.7.

Music application controlled by sensors

1. The first steps are the same as in the last example; the needed components are selected from the appropriate category shown in the tree view and arranged on the workspace. The music application proxy accepts boolean values for simple commands like "play/stop", "pause", "next track" and "previous track". The values for the volume are given by integer values ranging from 0 to 100. The mapping problem that arises is that the sensor board posts integer values whereas the music application accepts integer and boolean values.

Selection of entities

2. A direct connection would result in a type mismatch except for the volume parameters.

Partially valid connections

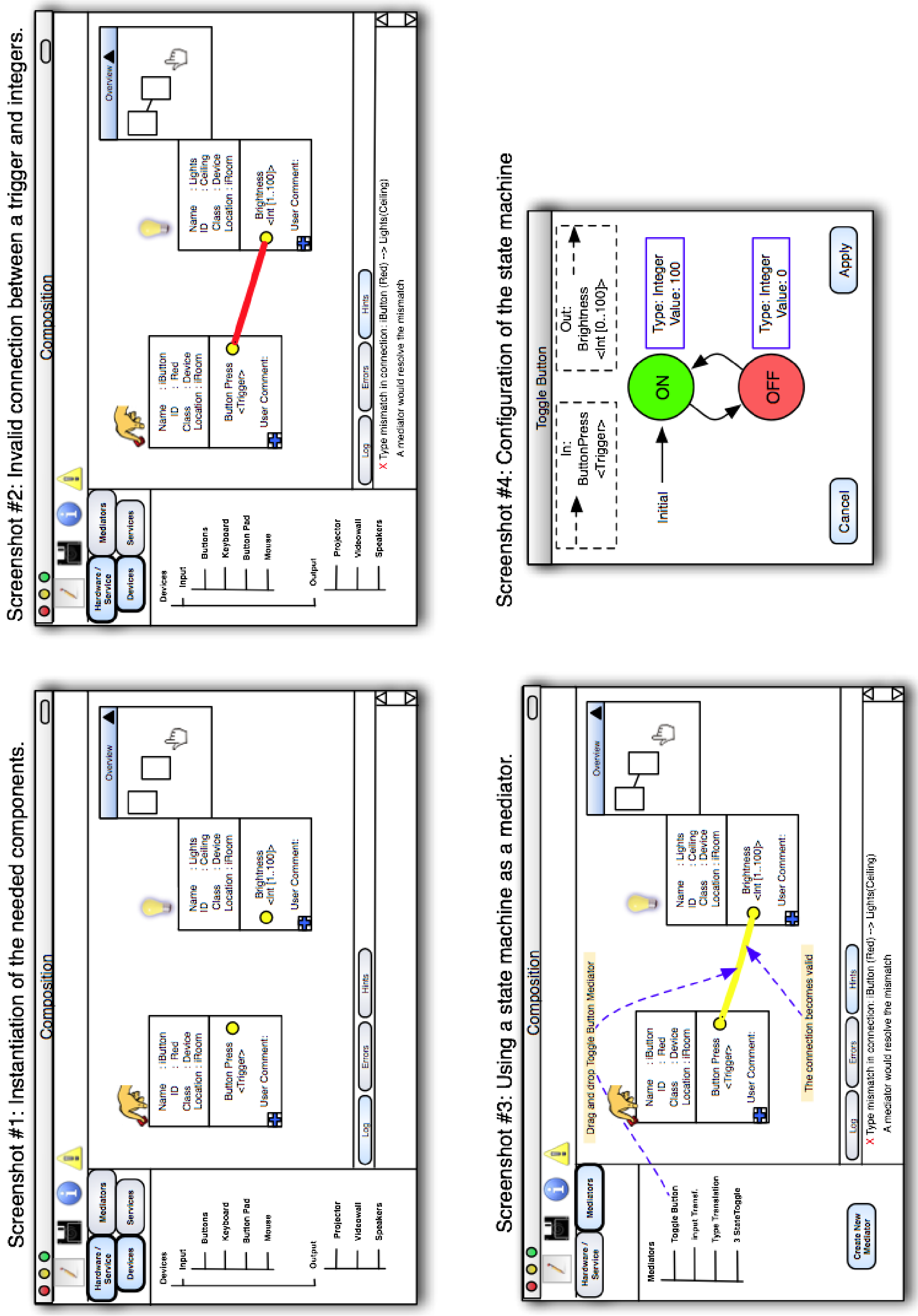


Figure 4.3: The storyboard for the example described in section 4.2.2.

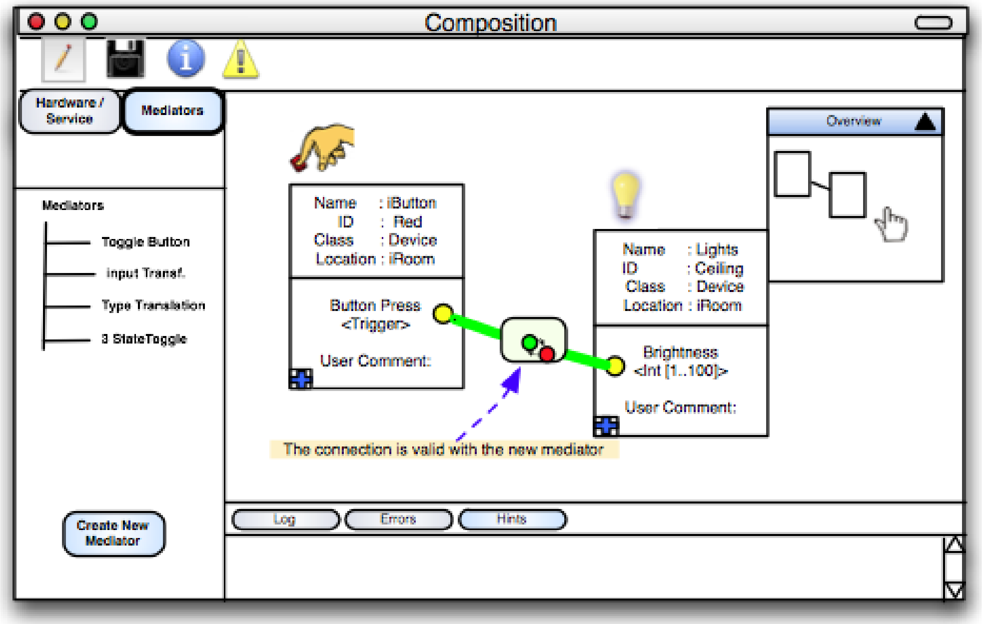


Figure 4.4: The final result for the example described in section 4.2.2.

3. To resolve the mismatch, conditionals are introduced that accept integers and yield out booleans. For the transformation of the integer values in order to match the volume scale of the music application proxy, a mathematical transformation is applied such that the incoming values are normalized in the range between 0 and 100.

Mediating conditionals

4. An arbitrary mathematically correct formula can be entered in the configuration field of the mediator. In the case that it is collapsed, a double click onto the component or a click on the cross unfolds the content.

Mathematical transformations

5. There are no type mismatches left in the final composition. In order to adjust the thresholds that control the sensor inputs, the conditionals can be altered at runtime and the results can directly be seen.

Final setup

This prototype can easily be extended at runtime by connecting different devices to the already connected ports or

Extensible at runtime

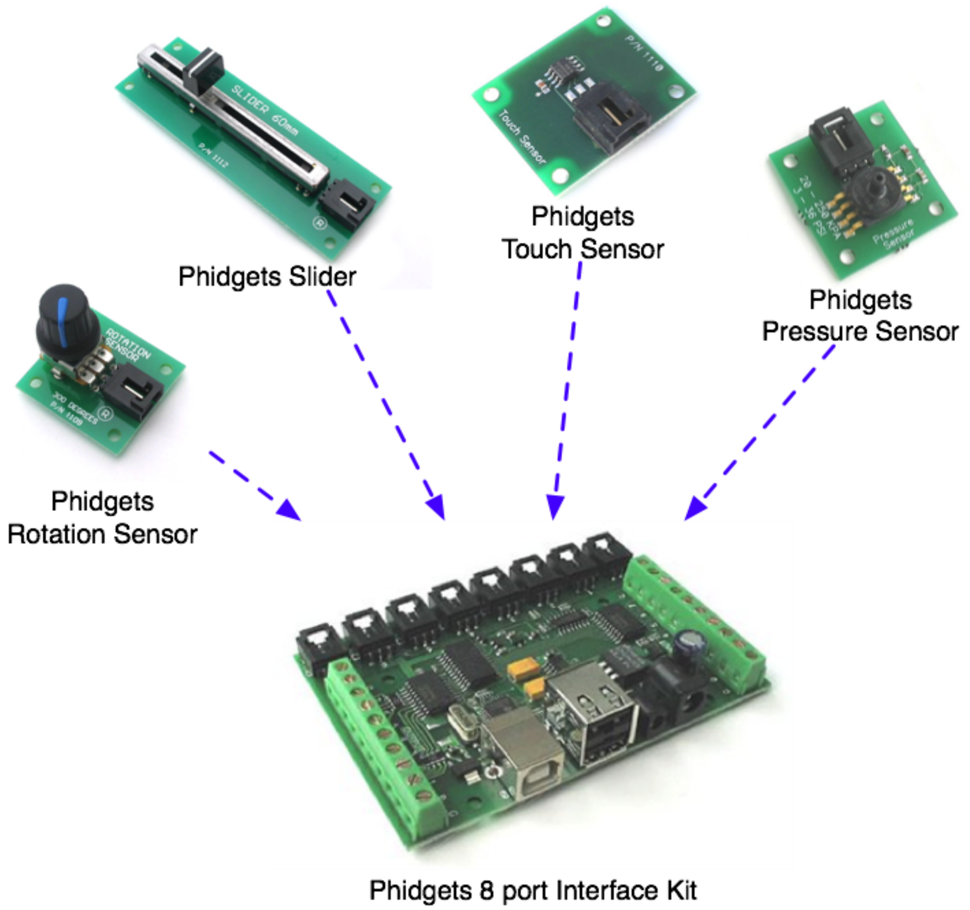


Figure 4.5: The set of Phidgets sensors that should be used for the scenario described in section 4.2.2. The different sensors are connected to one of the inputs of the Interface Kit sensor board. The sensor values lie between 0 and 999.

to the ones that are currently not connected. The rearrangement takes place at runtime as well as changes made to the threshold calculations.

4.2.3 Prototype evaluation

Comparison to Quartz Composer

The storyboards were extended to larger scales but not all of them are part of this work since only a general feel for the planned Patch Panel GUI should be provided. The decision of extending the Quartz Composer should be justified since

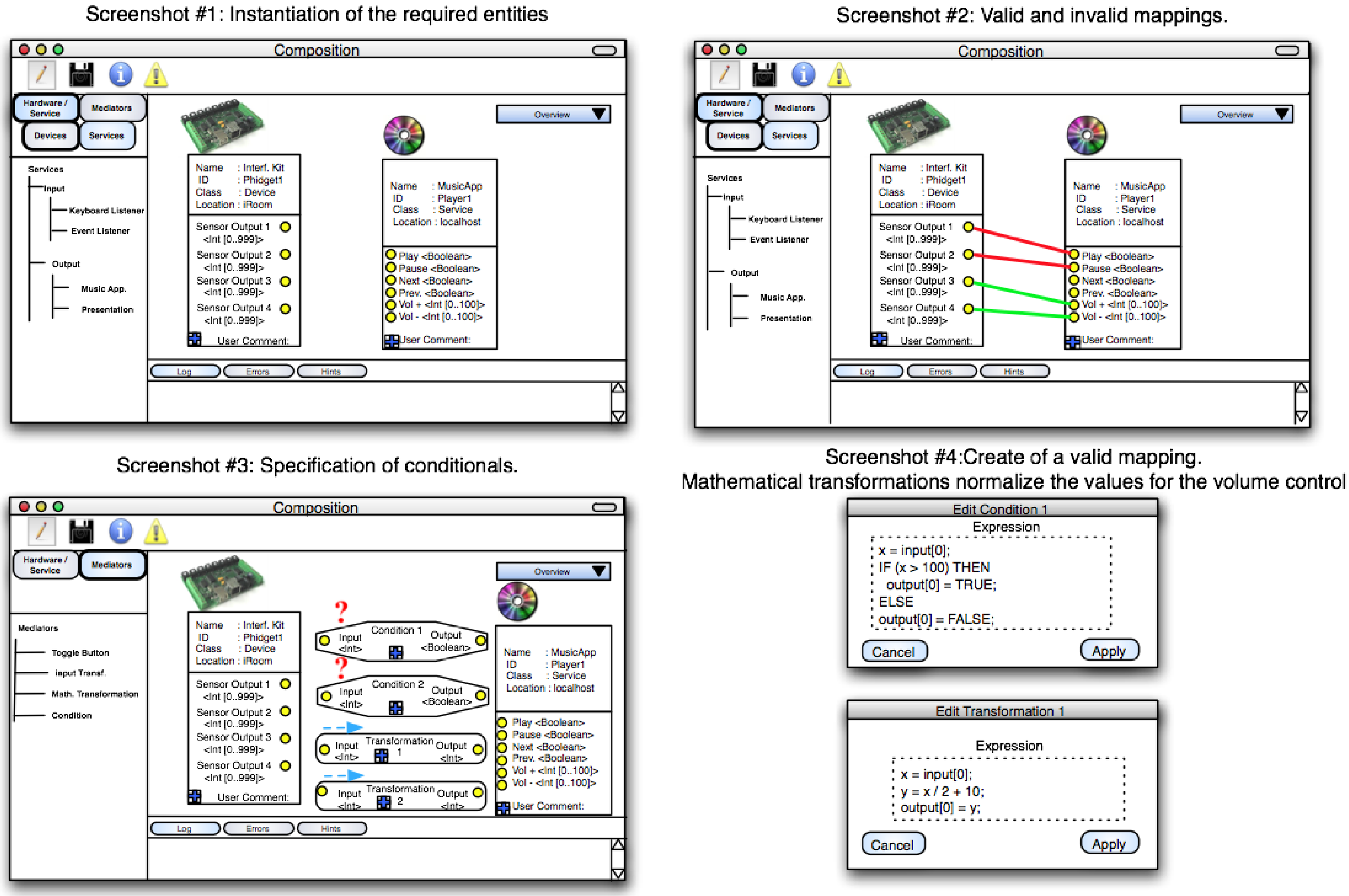


Figure 4.6: The storyboard for example described in section 4.2.2.

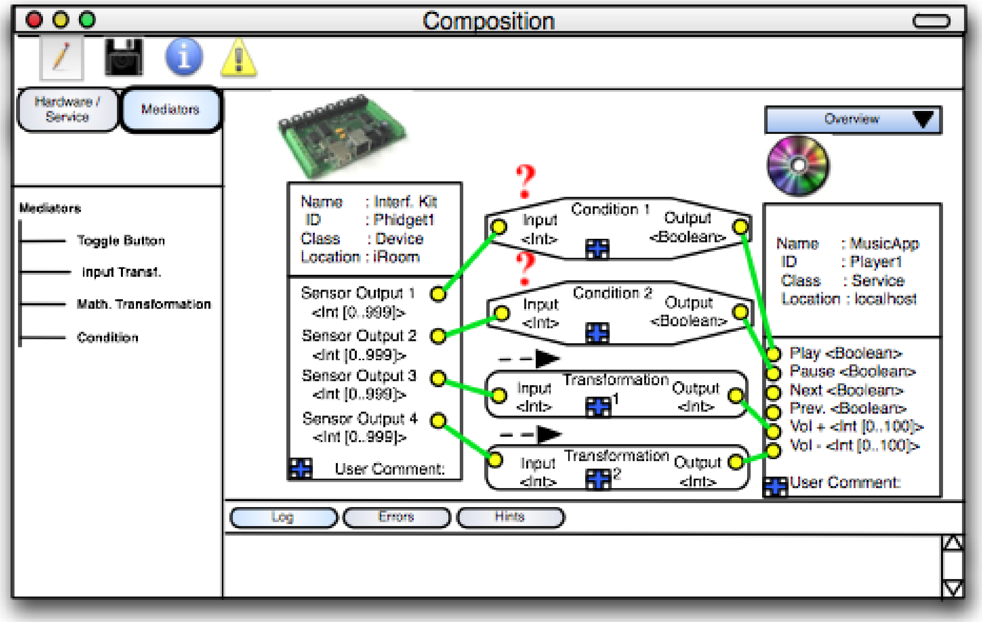


Figure 4.7: The final result for the example described in section 4.2.2.

that application incorporates many of the desired features extracted from related work and derived from early prototyping.

Evaluation revealed good concepts but also some usability problems

Demonstrating the storyboards to different persons turned out that the general concept of dragging and dropping components, connecting entities and configuring special functionalities inside an entity was plausible. Problems arose when devices and services had to be selected. Users complained that the tabbed pane was too hard to use in order to quickly find an iStuff component. Other questions concerning the interaction principles came up, especially at the point when state machines should be edited: Reviewers were afraid that the overview might suffer. One problem might be the issue of connecting single states to different entities because they initiate another event to be sent.

Striking similarity

After collecting first feedback and comparing the feature list of existing GUIs as discussed in chapter 3 it was found out that Quartz Composer provides a lot of the desired features listed in section 4.1. A way to extend the set of provided functionalities inside Quartz Composer was found by searching the web and experimenting with the knowledge available about the Quartz Composer classes.

Quartz Composer extension

The UI concepts described in the paper prototypes overlap to about 60 percent with the concepts provided by Quartz Composer. Thus, the decision was made that the extension of the existing application would hold more benefits for the iStuff project than a complete redesign of a graphical user interface for the Patch Panel. Instead of redesigning (yet another) graphical user interface, the iStuff project should be supported with a good GUI concept where the iStuff components and prototyping utilities could directly be integrated.

Growing popularity

Another argument for using Quartz Composer is that it gains more and more popularity in the design community. Thus, by reusing a familiar framework the need to learn a new tool is reduced.

Animations for ubicomp scenarios

Since Quartz Composer is originally designed to create interactive 3D animations it makes incorporating these animations into ubicomp interfaces trivial. Large public display scenarios as described in Ballagas et al. [2005] are one example or such a scenario.

Focus on extending Quartz Composer

Therefore, the focus of this work changed towards extending the Quartz Composer application and integrating a subset of the iStuff components that proof the idea's applicability. The general extendiblility for future iStuff components is pointed out in chapter 5

Earlier exploration than planned

Prototyping support is also provided by new integrations as well as by ideas described for future work. With the Patch Panel GUI as an extension of Quartz Composer, the prototyping capabilities of the iStuff toolkit can be explored much earlier than originally planned.

Limitation to Macintosh platforms

The disadvantage of that strategy is that with a Patch Panel for Apple Macintosh systems, the platform independence for the Patch Panel GUI is lost. However, because the department and most of its research work is based on Apple hardware, the decision was supported because the most important aim is to explore the prototyping capabilities of the iStuff toolkit and not to provide a platform independent prototyping suite. The iStuff project was created for research issues and not for commercial interests where this restriction would play a significant role.

Examination of Quartz Composer

The next chapter examines the Quartz Composer application and analyzes the integration possibilities of the new Patch Panel GUI.

Chapter 5

Quartz Composer as the Patch Panel GUI

Quartz Composer features

This chapter takes a closer look at Quartz Composer as an extensible candidate for the Patch Panel GUI. Therefore, its capabilities and features are presented in detail. A table comparing already existing and desired features summarizes the presentation. It is shown that the most important features are already covered by Quartz Composer and thus the decision for extending this application as a Patch Panel GUI can be justified. A collection of concepts that still have to be integrated in the scope of a Quartz Composer extension is shown in the final section of this chapter.

5.1 A closer look at Quartz Composer

Atomic functionality

Quartz Composer possesses a variety of built-in components which are suited for supporting the user constructing graphical applications like screen savers or real time animations, partially based on input data coming from different sources. Because of this application field most of the components that build the connection from the application to the operating system are designed for making calls of Mac OS X's graphics engine *Quartz*. Each component regarded on its own only provides atomic functionality like drawing backgrounds or sprites on the screen. Other methods like receiving inputs from the keyboard, counting, evaluating conditionals or performing mathematical transformations are also available. Thus, the *combination* of many different components makes the application extremely flexible and powerful in terms of design freedom. A very similar concept is applied to other application fields (cf. chapter 3) and is especially one of the design keys of the iStuff toolkit.

Data-flow metaphor

Each Quartz Composer component, in the following referred to as *patch*, provides, depending on its type (see below), different input and output ports that represent its parameters. Input may be gained from system components like mouse and keyboard or from transformations based on time (like counting or interpolation). External sources like image libraries, RSS feeds and image files are supported as well. Since data is generally provided by outputs of different patches one could talk of a *data-flow metaphor*; data comes in at one point and traverses the composition until it is consumed by some patch that processes the data in order to initiate a system (in the original intention graphical) routine. To keep up the data-flow metaphor and the natural understanding of the application concept, output ports can only be connected to input ports.

5.1.1 Types of Quart Composer patches

Types of patches (components)

As already mentioned , there are three different types of patches (cf. the Apple Quartz Composer Tutorial[1]) that determine its behavior and when it is executed. Figure 5.1 shows examples for the different types of patches:

Data sources

- *Providers (blue label)*: This class of patches provides data retrieved from external sources such as mouse or keyboard. They receive their data from external sources and set their output ports accordingly.

Data manipulation

- *Processors (green label)*: These patches take data from the input ports or from internally specified sources (see the "Image Importer" patch) whenever one of the inputs changes. Patches of this type process data

[1] http://developer.apple.com/

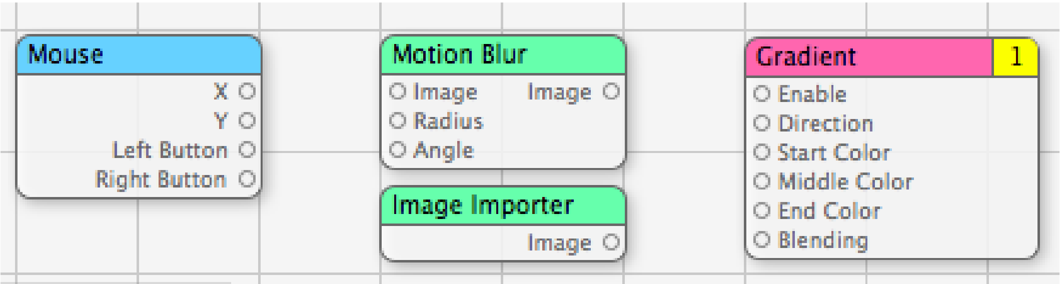


Figure 5.1: The different types of patches: The leftmost "Mouse" patch provides data sent by the mouse attached to the system. The two middle patches are examples for processors, one of them having both, input and output ports, and below a processor that only possess an output ports since it provides an image loaded from the hard drive. The rightmost patch is a consumer that renders a gradient on the screen. Its appearance depends on the input parameters.

at specified intervals or as soon as one of the input values changes.

Rendering to destination

- *Consumers (purple label)*: A consumer renders its input to a destination. In the Quartz Composer context, this means that routines inside the operating system's graphics engine are called such that something is rendered to a screen, for example. This class also provides a boolean"Enabled" parameter that activates or deactivates the execution of the patch. The number in the upper right corner of a consumer patch determines the *execution order* of a patch. This order may become important when different graphical components have to be drawn after each other. An example would be two patches from which one draws the background and the other one renders an object on the screen. In order to gain a correct result, it is important that the background is drawn before the shape in the foreground. The other way round, the background would cover the shape. The order of the consumer patches' execution can be modified manually. Changes affect the general order meaning that the other numbers are set according to the one changed.

5.1.2 Patch configuration

Input ports are set by connected outputs ports or manually

As already described, input parameters are influenced by values coming from system sources or from input ports to which another output port is connected. For system sources, there is no way of directly changing values that are sent to a patch. For input ports, however, that are not connected, values can be set by the user by double clicking the port and manually entering the desired value. As soon as the manually set input port is connected, the fixed value is replaced by the one provided by the connection.

Inspector window

Another way of changing inner settings of a patch is provided by an *inspector window* that holds general information about a patch like its name, description or an optional user comment as well as specific information individual for each patch. Figure 5.2 shows the three panes of the inspector view of the "Math" patch. The "Information" and the "Input Parameters" panes are always shown although there are patches that do not provide input ports. In this case, the pane simply remains blank. For the shown patch, a formula can be specified by selecting operations from combo boxes and either setting values manually or taking them from input ports. Mixing these options is possible, too. In figure 5.2, the patch's first input port is connected to another output port whereas the second input value is specified by the user. This is the reason why the first line is grey indicating that it is not accessible by the user. Other patches let the user adjust the inputs by providing scroll wheels or checking boxes if the input types are boolean.

The "Settings" pane is optional

The optional "Settings" pane provides little individual user interfaces that let the user configure patch-specific options or behavior. The patch shown in figure 5.2, for example, lets the user set the desired amount of input ports dynamically.

5.1.3 Grouping and abstraction

Macro patches

Sets of patches can be grouped into a *macro patch* that abstracts from the view onto several patches to one single patch. Ports that have to be available outside the ab-

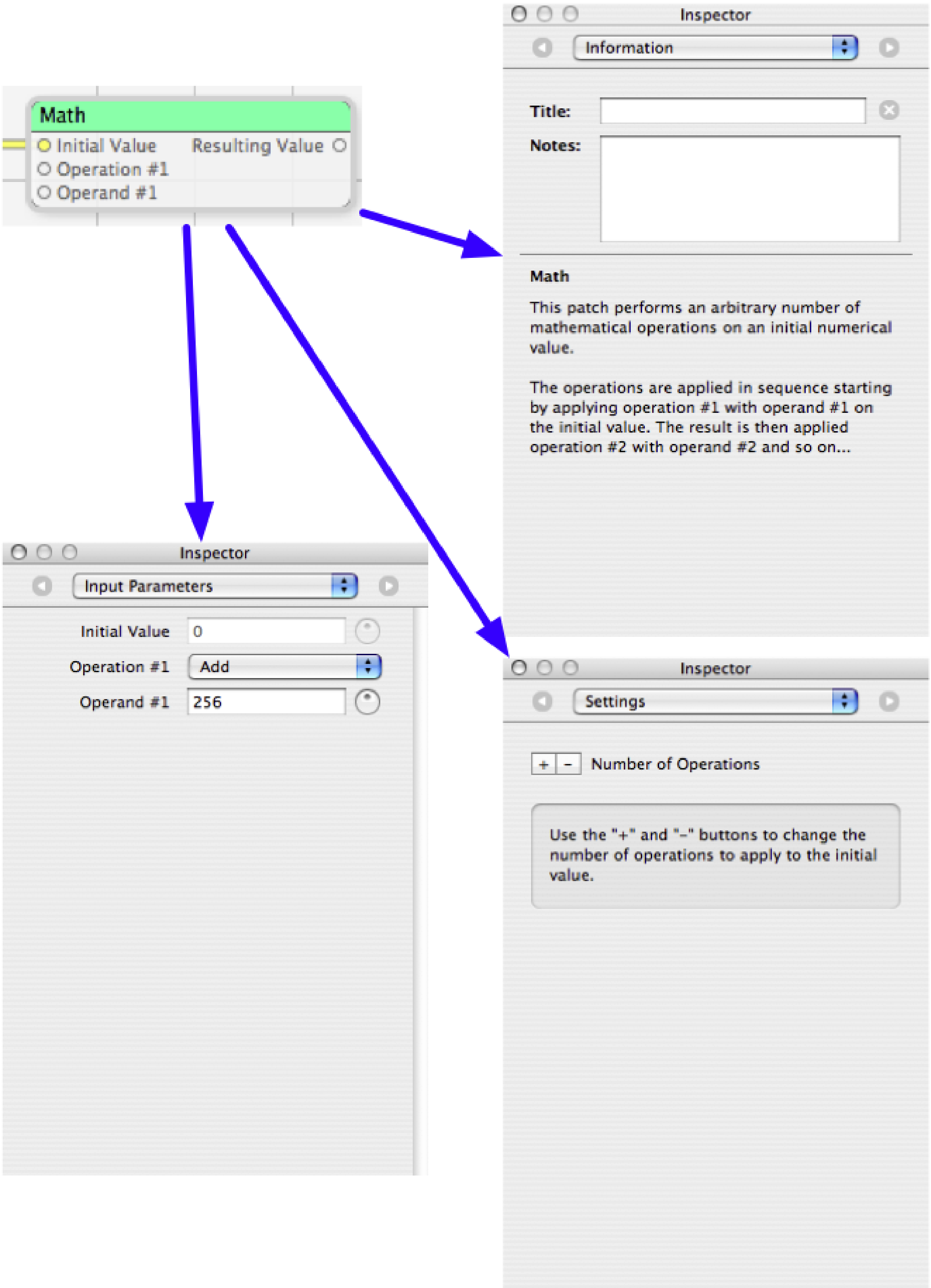


Figure 5.2: The three different inspector views of a "Math" patch. The Information pane shows general information about the patch, the second pane allows manipulation of the input ports (whereas the first input is connected and therefore no manipulation is allowed). The third panel lets the user adjust additional settings, in this case the number of operands needed.

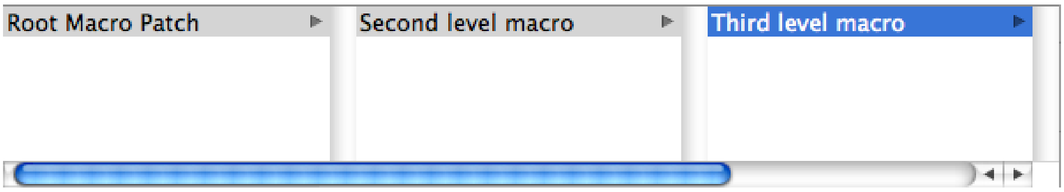


Figure 5.3: The Hierarchy browser helps the user navigate through the hierarchy created by macro patches.

straction can be published and renamed in order to clarify their meanings. This encapsulation is repeatable such that macro patches themselves can again contain other macro patches. The evolving hierarchy is visualized with the *hierarchy browser* shown in figure 5.3. With its help, users can navigate through the different abstraction layers similar to Apple Finder's column view.

Distinction

Macro patches can be distinguished from standard (atomic) patches by their shape; the first have rounded corners whereas the latter appear as rectangular shapes. A double click on a macro patch or making use of the hierarchy browser to navigating through the abstraction levels reveals the inner structure of a macro patch abstraction. Figure 5.4 illustrates the summarization of different patches into one macro patch where two input ports and one output port are published from the inside in order to be available for the rest of the composition. Macro patches appear according to their inner patches that means if only processor patches are grouped together, the abstraction also appears in the same color which is green in this case. Otherwise, if different types of patches are intermingled, the following hierarchy is applied: Processors $\ll$ Providers $\ll$ Consumers whereas "$\ll$" means that the right part of the relation determines the macro patch's appearance.

5.1.4 Finding and instantiation of patches

Incremental search

To select a patch, an incremental search lets the user type in a part of the patch's name or description and select from a pre-filtered list of results corresponding to the search keywords. The results can be ordered by their name or category in an ascending or descending way.

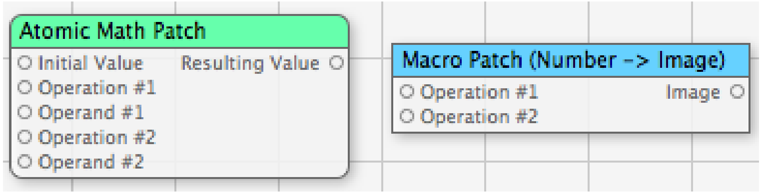


Figure 5.4: An atomic "Math" patch is shown on the left. On the right, a macro patch consisting of different patches from which three ports were published such that they are available at the current abstraction level.

Patches instantiation

Dragging the name of the patch into the composition area lets the user directly specify the location where the new instance is created. A double-click creates the new instance in the center of the composition area. The keywords for the search can be part of a patch's name, its category or of its description which is displayed below the search window. Figure 5.5 shows the search results for patches that have to do with displaying issues.

5.1.5 Automatic type checking and conversion

Type checking

Whenever ports are connected, Quartz Composer automatically checks the connection's direction and the parameter types the user tries to connect. As already mentioned, connections can only be drawn from output to input ports, the opposite direction is not possible. The application even just ignores the user's mouse gestures.

Different connection colors

When port types are to be connected that match, Quartz Composer indicates this by drawing the connection in yellow color. If the types do not match at all, for example, when boolean values should be connected with a patch that only accepts images, the connection is drawn as a white line and disappears as soon as the mouse button is released. Whenever a valid connection is found during the connection process, this is indicated by a yellow or orange line. The latter indicates that data types that were connected are not the same but a conversion can be applied.

Connection example

For example, a user might want to connect an output port

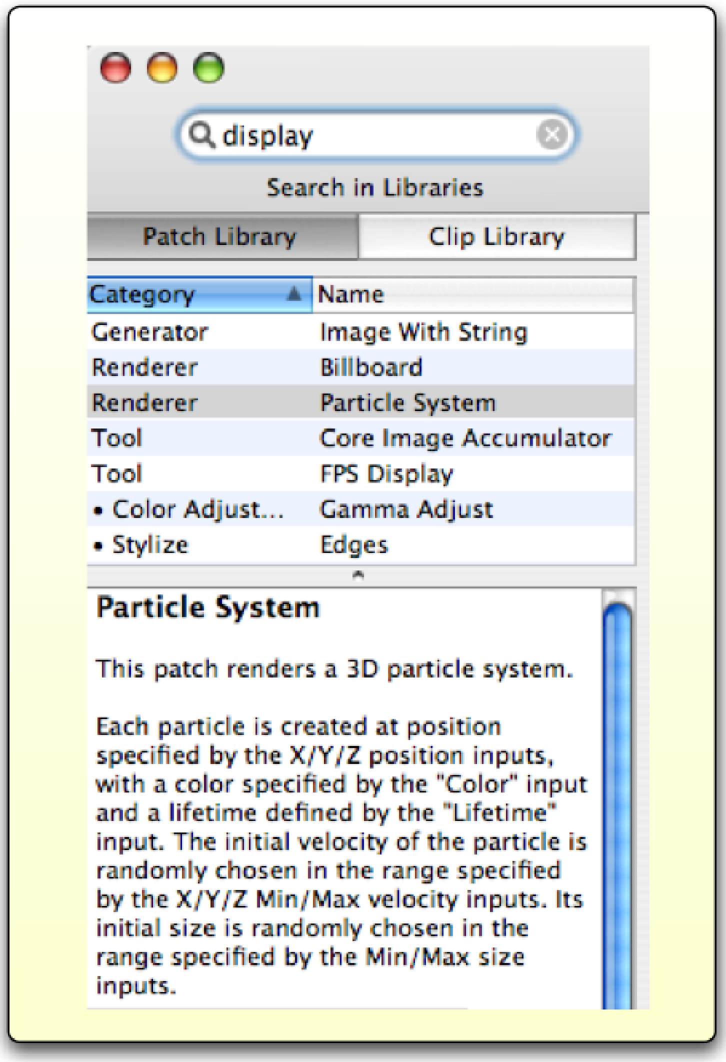


Figure 5.5: The incremental search starts as soon as characters are entered in the search field. The patches' names, categories and descriptions are searched.

of type integer to an input port that accepts boolean values. The connection is drawn in orange color and interpreted in a meaningful way by the application, namely that any value greater than zero corresponds to TRUE, and FALSE otherwise (cf. figure 5.6). The same works with integers connected to strings, for example.

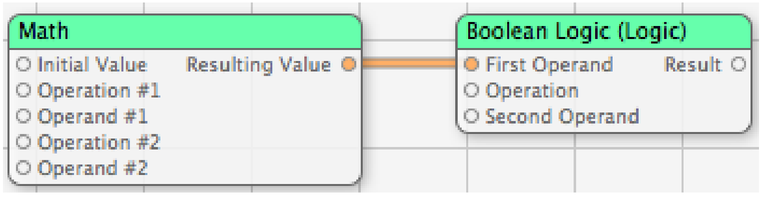


Figure 5.6: Integer values coming from the "Math" patch should be processed by the "Boolean Logic" patch that accepts boolean values. Instead of rejected the connection, incoming types are interpreted as TRUE if they are greater than 0.

Non-reliable conversion

Red lines indicate that a connection can be used but the values passed cannot be interpreted in a correct way and no type conversion is applicable. This is the case when structures get connected to strings. The only information available to a string port is that the input is of type "QCStructure" which interpreted as a string. Figure 5.7 shows an example of this special connection situation.

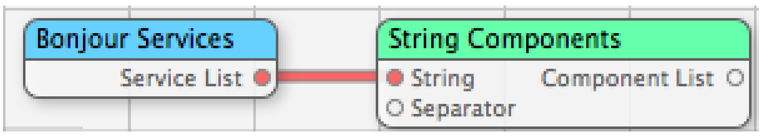


Figure 5.7: When structures get connected to string inputs, the only information available for the input is the string "QCStructure".

5.1.6 An example

QC functionality illustration

After the different Quartz Composer parts are described in more detail, an example shown in figure 5.8a should illustrate the features described so far.

Visualization of mouse coordinates

The mouse attached to the computer shall provide the data needed for the composition. In order to show the values yielded out by the "Mouse" patch, a blue gradient background is drawn by selecting the "Gradient" patch and specifying the color values with the inspector. Onto the background, two sprites are to be drawn, one displaying the string "Mouse X" and the other one showing the values coming from the X-output port of the "Mouse" patch.

Implicit transformation

Since sprites only accept inputs of type "Image", an intermediary patch "Image With String" has to be introduced. This patch takes the number values coming from the "Mouse" patch output, automatically transforms them into a string value whereas the resolved type mismatch is visualized by an orange instead of a yellow connection, and sets the incoming string as an image at the output port.

First abstraction

The positions of the two sprites can be set directly by manipulating the corresponding input ports. Since the patches responsible for drawing the label are not important to be shown, they are summarized in a macro patch named "Mouse X" (cf. figure 5.8b).

Second abstraction

In order to show different levels of abstraction, the two patches providing the visual feedback of the mouse coordinates were once more encapsulated into a macro patch called "Background" (cf. figure 5.8c). The input port "X Position" of the "Sprite" patch inside the macro patch was published such that it is still accessible from the root. The graphical output stays the same with all three abstractions and is shown in figure 5.8d.

5.2 Functionality missing for the iStuff project

Desired features vs. already provided ones

After the detailed description of the functionality provided by the Quartz Composer application, table 5.1 should summarize the features already provided by Quartz Composer compared to the desired feature list in chapter 4.

Quartz Composer provides a very useful basis for the Patch Panel GUI

Since about 60 percent of the feature list are covered solely by Quartz Composer, the iStuff project group decided to augment the existing application instead of redesigning a very similar graphical user interface. The information to extend Quartz Composer was retrieved from custom discoveries as well as from searching the web. Among others

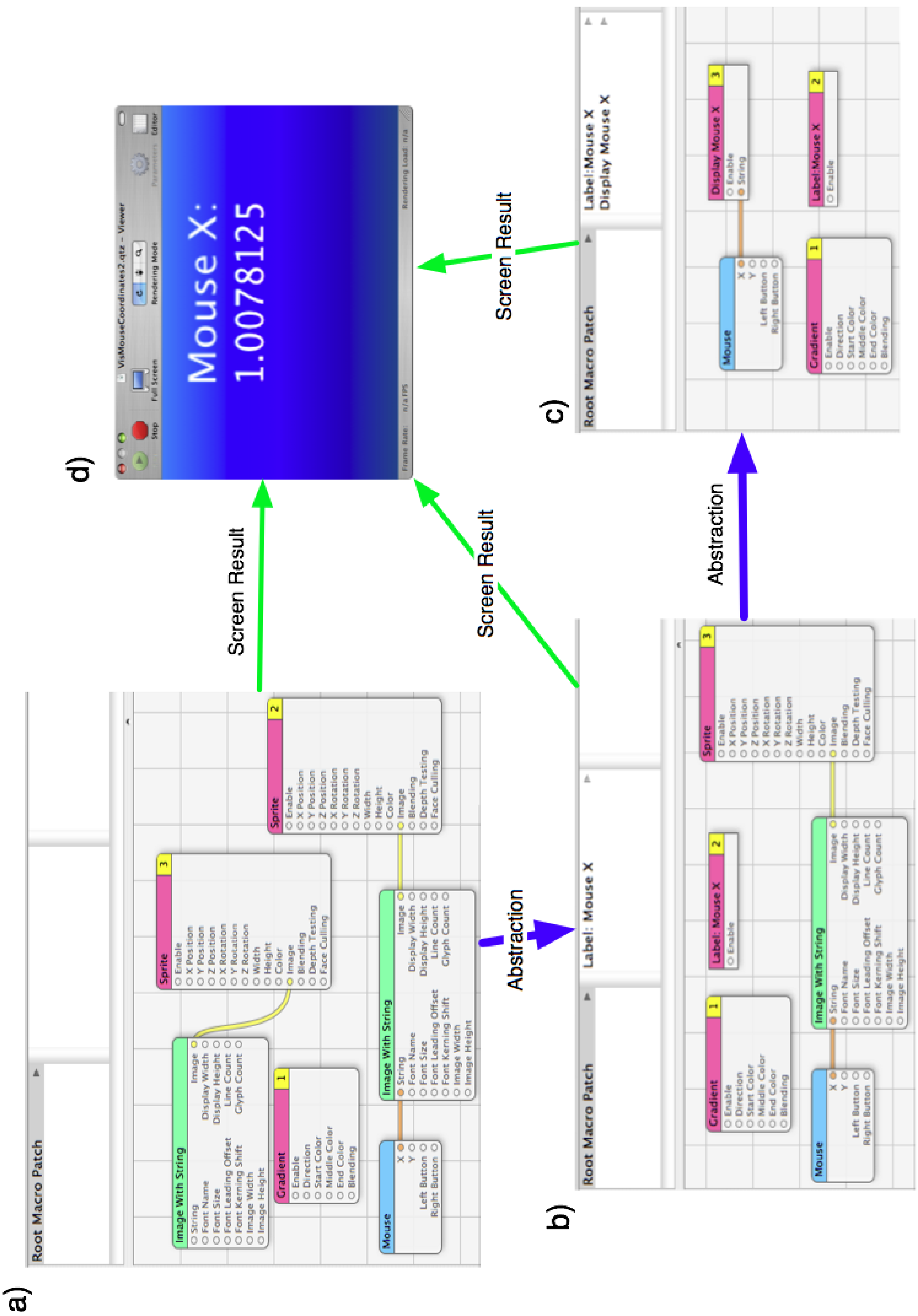


Figure 5.8: The example described in section 5.1.6. The graphical output stays the same throughout all three abstractions (a-c) and can be seen in d.

Desired feature	Available in QC	On feature list
Composition	Yes	-
Easy retrieval of components	Yes	-
Drag&drop support	Yes	-
Avoidance of illegal connections	Yes	-
Automatic type conversion	Yes	-
Consistent flow of information	Yes	-
Instant application of changes	Yes	-
Abstract representation of entities	Yes	-
Provision of template values	Yes	-
Abstract testing	Partially	Partially
Highlighting of current selections	-	Yes
Occlusion avoidance	-	Yes
Panning and zooming	Panning	Zooming
Abstraction	Yes	-
Overview window	-	Yes
List of currently running proxies	-	Yes
Iconic representations	-	Yes
Generation of events/values	-	Yes
Graphical visualization of values	-	Yes

Table 5.1: Desired features for the Patch Panel GUI vs. features already implemented by Quartz Composer

a web log called Clockskew[2] provides useful information on how to start with a custom plugin (although not all information is correct).

Functionality to be added for the iStuff project

The next step in the GUI development was to outline features and functionalities that had to be added for the Patch Panel GUI. In the following, they are described whereas some of them were realized as independent applications to be launched in parallel to Quartz Composer.

5.2.1 Integration of iStuff components into the GUI

Subset of iStuff components

A subset of the existing iStuff toolkit components should be added in order to support the rapid prototyping process and to serve as a proof of concept. The subset should include sensor kits (e.g. Phidgets, SmartIts), software prox-

[2] http://www.clockskew.com/blog/?p=15

ies (e.g. presentation software and sound card controllers) as well as parts from the iStuff mobile toolkit (cf.Memon [2006]).

Support for additional off-the-shelf-hardware

Extensible software framework

Available hardware like sensors, motors, buttons, speakers or remote controls for which the iStuff proxy strategy can be applied (cf. chapter 2) should be easy to integrate into the Patch Panel GUI as it is intended by the original iStuff proxy concept. For this purpose, an object-oriented software framework has to be created such that a lot of development is saved and thus new components can quickly be integrated into the framework.

Evaluation support

Data visualization

Visualization aids like a plotter or windows displaying several different information provided by output ports should help the developer to see the progress of values passed into or coming out from certain patches.

Support for custom extensions

Integration of non-iStuff patches

Additional patches, like the "Filter" patch mentioned in the last item, that do not correspond to an iStuff component but which facilitate the prototyping work should be provided. The software framework should be designed to be open to future work such that new patches can quickly be integrated.

Event debugging support

Event logging

The events posted to the Event Heap including their fields should be accessible to the developer. Since this application already exists in form of the *Event Logger* as part of the iROS distribution, it was decided to leave it as an external application because it provides a good mechanism that can also be used in contexts where the Patch Panel GUI is not needed. An integration would create unnecessary overhead and wouldn't hold additional benefits for the whole iStuff project.

GUI support for proxies

Quickly setting up proxies

The different proxies for the iStuff components were originally started via the command line. An external GUI should be developed that wraps around the command line and allows the developer to quickly set up a proxy and launch it from that GUI. This kind of application does not fit into the Patch Panel scope but is needed for the rapid prototyping process. Therefore it is part of this work but left outside the Patch Panel GUI.

5.2.2 Final comparison

Planned extension of Quartz Composer

After comparing existing features in Quartz Composer against the desired feature list for the Patch Panel GUI (cf. section 5.2) and planning what features are to be added in the scope of this work, table 5.2 summarizes what features are already available, planned to be integrated and which ones are scheduled for future work.

Postponed features

Some of the originally desired concepts like the overview window, connection bending, labeling or iconifying, autoscrolling or zooming were left out at this point because there is no documented API for the Quartz Composer editor at the moment. Integration of these functionalities would require a more detailed insight into the existing framework than available at the moment. Because of the missing Apple support, this work has to be postponed until more information about the Quartz Composer framework is available. The features left out can be classified as "additional" features that support the interaction with the Patch Panel but that are not absolutely necessary for the proof of

Desired feature	Available in QC	To be added	Postponed
Composition	Yes	-	-
Easy retrieval of components	Yes	-	-
Drag&drop support	Yes	-	-
Avoidance of illegal connections	Yes	-	-
Automatic type conversion	Yes	-	-
Consistent flow of information	Yes	-	-
Instant application of changes	Yes	-	-
Abstract representation of entities	Yes	-	-
Provision of template values	Yes	-	-
Abstract testing	Partially	Partially	Partially
Highlighting of current selections	-	-	Yes
Occlusion avoidance	-	Yes	-
Panning and zooming	Panning	-	Zooming
Abstraction	Yes	-	-
Overview window	-	-	Yes
List of currently running proxies	-	Yes	-
Iconic representations	-	-	Yes
Generation of events/values	-	Yes	-
Graphical visualization of values	-	Partially	Partially
New features described in this chapter			
Integration of iStuff components	No	Yes	-
Support for additional HW	No	Yes	-
Evaluation support	Partially	Partially	Partially
Support for extensions	No	Yes	-
Event debugging support	No	Yes	-
GUI support for proxies	No	Yes	-

Table 5.2: Overview: Features available in Quartz Composer, newly integrated ones during this work and features scheduled for future work

concept this work follows. Even if only a subset could be integrated in the future, the Patch Panel GUI in the version presented here already suits most of the needs of the iStuff project

5.2.3 Benefits and disadvantages

Future realization

Although this leaves out some of the initially intended concepts, the iStuff project group assumes that with later versions of Mac OS X, maybe even already with version 10.5, developer support will increase.

Loss of platform independence vs. advance in research

On the one hand, the decision for the Quartz Composer as a basis for the Patch Panel GUI has the disadvantage of loosing the platform independence most of the iStuff components had since they were written in JAVA. On the other hand, extending Quartz Composer as the Patch Panel GUI means a large step forward for the iStuff project that can be taken earlier than planned. This is a very important benefit for the undertaken research work performed at the department.

Rest of independence is kept

Another argument is that the rest of the existing platform independence is kept. Only the Patch Panel application has to be run on an Apple Macintosh system. However, this does not break the communication model of the Event Heap structure because the iStuff core functionalities like the proxies setup or the event logging are separately available.

Chapter 6

Extending Quartz Composer as the Patch Panel GUI

Patch Panel GUI extension

The development platform for the Patch Panel GUI is set and the features that are to be integrated are clarified as well. This chapter describes how these features were incorporated into Quartz Composer. Additional Quartz Composer patches as well as external applications that support the prototyping process are presented afterwards. At the end of this chapter, implemented examples are shown in order to illustrate the cooperation of all the utilities developed in the scope of this work.

6.1 Integration of the Patch Panel into Quartz Composer

Integration of custom patches

Ways to write custom plugins for Quartz Composer were found by analyzing the Quartz Composer program structure and searching the web. One very interesting entry is to be found in the Clockskew weblog[1] where a good, although not completely correct guidance is given. Figure 6.1

[1] http://www.clockskew.com/blog/?p=15

shows a conceptual class hierarchy of a Quartz Composer patch (QCPatch).

6.1.1 The iStuff Patch hierarchy

Subclasses

So starting with the discovered information, subclasses were created from the QCPatch class in order to implement custom patches. Since one demanded feature is the easy extensibility of iStuff components and the patch library, respectively, an object oriented framework was created that allows subclassing from different specializations of the QCPatch class, according to the current needs.

New custom patch

Thus, in order to create a custom patch with new functionality, an *iStuffCustomPatch* can be subclassed. If a patch is needed that registers for special events on the Event Heap or post events onto it, the *iStuffProviderPatch* and *iStuffConsumerPatch* class should be subclassed, respectively. User interfaces in form of NIB files [2] for the "Settings" pane shown in the inspector window of a Quartz Composer patch are also already implemented but can be modified if needed. The files implementing the user interface are subclasses of the *QCInspector* class (cf. figure 6.1).

Connection management

Each iStuff patch class already provides methods for the Event Heap connection management. The *execute* method is called with every execution cycle as well as *initialization methods* that are run whenever an instance of a patch is created in a composition. Methods that are needed for the Patch Panel functionality can be added for each custom extension.

Decoupled threads

For each iStuff component, a connection to an Event Heap (specified in the "Settings" pane) is established. Every *iStuff provider patch* decouples an own thread that waits for events of the appropriate type to appear on the Event Heap. The thread is started as soon as a connection has been es-

[2]NIB files are responsible for the appearance of the user interface of an application. Apple's Interface Builder (cf. section 1.1) is an easy-to-use and very powerful tool to rapidly create a graphical user interface and its underlying functionality. The model of the user interface is implemented in accompanying Objective-C files.

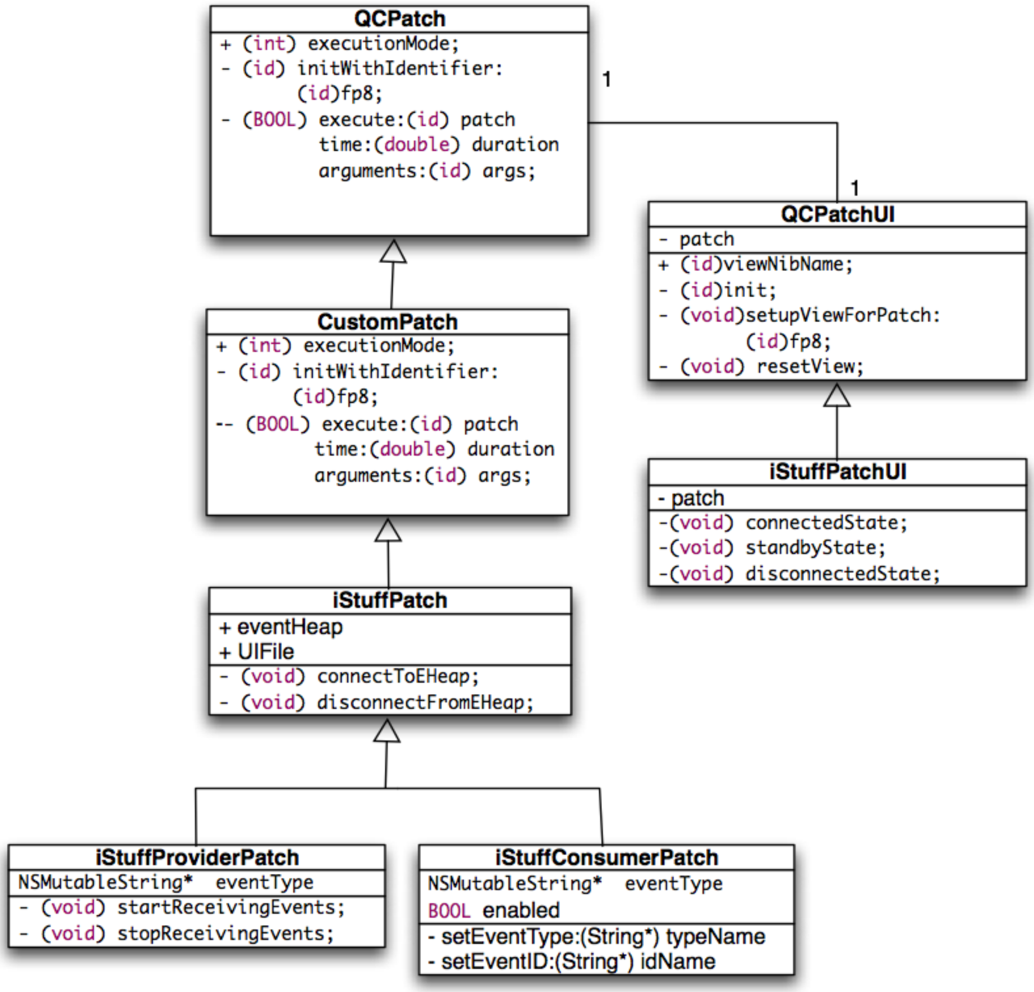


Figure 6.1: The basic patch class hierarchy. From a *QCPatch* class, custom patches can be derived. Its ancestors are *iStuffPatch* classes that already provide mechanism to connect to an Event Heap. They can be specialized to *iStuffProviderPatches* to register for events and *iStuffConsumerPatches* to post events on the Event Heap, respectively.

tablished. *iStuff consumer patches* do not launch separate threads but only post events of the specified type to the Event Heap depending on their implementation.

Source code online

The complete source code of the whole framework and the implemented extensions are available for download at the berlios iStuff project site[3] . A detailed guide on how to write

[3]http://developer.berlios.de/projects/istuff/

custom patches goes beyond the scope of this work will be retrievable on the iStuff project site[4] in future.

6.1.2 Managing connections

Automatic connection

By default, each iStuff patch connects to the Event Heap running on the local machine. Alternatively it takes the Event Heap the preceding patch connected. In case that a composition was saved and gets loaded, each patch tries to establish a connection to the Event Heap they were connect to at the time the saving took place.

Custom "Settings" pane

From the "Settings" pane inside the inspector window, the connection management details can be seen and modified (cf. figure 6.2). Here, the discovered Event Heaps are displayed and custom ones can be specified in case they were not correctly discovered. It is also possible to determine whether connection changes should only be valid for the current patch or they should be applied to all patches in the composition.

Event ID

Additionally, in order to be able to distinguish between different components of the same kind (e.g. two Phidget Interface Kits), an event ID can be specified. With each event a field with a value for the ID is sent besides others specified. iStuff provider patches can be configured either to register for *any* event of the corresponding kind or to register for events of the defined kind that *additionally* contain a specific event ID. iStuff consumer patches always provide such an ID and it depends on the proxies registered for an event type whether they check for it. The event ID is determined by the name of the iStuff patch.

6.2 Integrated iStuff components

Components developed during thesis

In the following, a list is presented that shows which iStuff components have been integrated during this work. Since

[4]http://media.informatik.rwth-aachen.de/istuff/

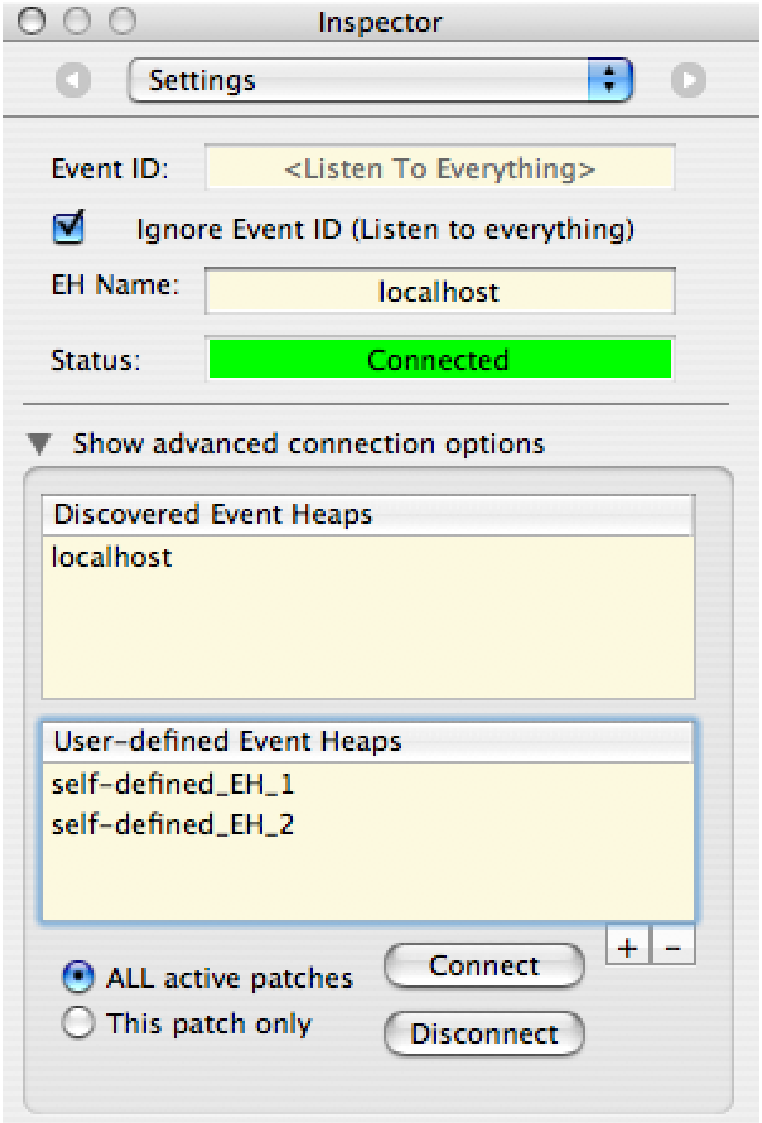


Figure 6.2: The "Setting" pane for iStuff patches reveals information about discovered Event Heaps and lets the user specify the event ID id desired (iStuff ProviderPatches only). In an extended view, the Event Heap connection settings can be changed and custom Event Heap names can also be entered for the case that the discovery was incomplete. The section at the bottom provides the choice of changing the connection settings for every patch or the current patch only.

a framework was developed (cf. section 6.1), the integration of new components is no longer very time consuming. iStuff components made available as a proof of concept for the Patch Panel GUI so far include:

- Phidgets Interface Kit (and thus all sensor connectable to it)
- Phidgets Accelerometer
- Phidgets RFID Tag Reader
- Phidgets Servo Motor Control (up to four servos)
- SmartIts sensor boards
- Mobile Phone Key Listener
- Mobile Phone Controller
- Presentation Controller
- Apple Powerbook Tilt Sensor [5]
- Keyboard Listener
- Listeners for the Sweep technique [6]
- Teleo Toolkit Input
- Teleo Toolkit Output

6.3 Support for the prototyping process

Other extensions

Besides the patches for the iStuff components, a number of additional patches were developed (or at least planned as extensions) in order to take over tasks always repeating like smoothening values or influencing the rate at which events are sent or processed. These patches are described below in more detail.

[5]Inside Apple Powerbooks of the latest generation, a tilt sensor immediately parks the hard drive when the laptop is rapidly moved or dropped. The sensor values can be read out by software.

[6]Ballagas et al. [2005] developed a techniques that allows the detection of two-dimensional movement of a mobile phone by processing the image data from the camera of the mobile phone. This approach was primarily used for large public screen interactions.

6.3.1 Filter (integrated)

Smoothening

The "Filter" patch provides inputs for numeric values and lets the user specify when a new input should be processed. For this purpose the user specifies at a second input or via the "Settings" panel the absolute difference the new and the old values must differ from each other. Is the absolute difference greater than the value specified, the new value is sent to the output port, otherwise the old output value remains. The filter patch is very useful for devices that send many events per time interval and whose values do not differ very much or if small changes are not relevant for the developer. An example for such an iStuff device is the Phidgets Accelerometer that sends events at an almost constant rate even if it is not moved.

6.3.2 Threshold (integrated)

Exact value comparison

This "Threshold" patch provides a boolean output which becomes TRUE if the input is below, equal or above a certain value. Although the "Conditional" patch already provides a very similar functionality, the values entered in the "Settings" panel of the Threshold Patch are not rounded. The "Conditional" patch does that when the values are not entered by double-clicking the input ports.

6.3.3 Buffer (future work)

Collect and lists input data

The "Buffer" patch can also be used for examination purposes. Strings or numeric values passed to it are buffered are shown in a console-like window. For each input, a separate window can be instantiated.

6.3.4 Plotter (future work)

Visualization of data progression

The "Plotter" patch visually supports the developer in following the development of values output by other patches.

Outputs from other patches can be passed into the corresponding input and are displayed in a separate window for each input type if desired. As there can be any arbitrary number of patches in Quartz Composer, developers could also connect any number of plotters to any patch. The inputs are drawn as a graph over time in an OpenGL context. A screenshot of the results can be taken for future purpose or the plotted values can be saved to a file.

6.3.5 Display (future work)

Permanent port value visualization

A separate window controlled by a "Display" patch is shown. The patch provides a number of inputs that are dynamically changeable at runtime. To that patch, booleans, strings and integers can be passed and their current values are displayed in labeled fields. The concept is similar to the template composition Quartz Composer offers when a new project is started. There, only one input is displayed for demonstration purposes. Since sometimes only current values of certain outputs are relevant, the "Display" patch would be an alternative to the plotter.

6.3.6 Help from built-in patches

Hidden patches

By enabling hidden patches in the Quartz Composer with the help of another plugin from the Clockskew website[7] , a number of patches can also be used for supporting the developer. Although those patches were not intended to be accessed by default, they also present a good addition to the built-in and custom patches.

Triggers and rates

The "Signal" patch, for example, provides a way to send a trigger at a certain time interval (similar to the "bang" in Max/MSP). The "SignalAndHold" patch keeps a certain value until it receives a TRUE trigger. This can also be used for patches that only send a value once and that would be lost otherwise.

Powerful patch: "Java Script"

Another patch that should be mentioned here is the one

[7] http://www.clockskew.com/blog/?p=14

with the most flexibility: The "JavaScript" patch. However, one should take care about not specifying too much functionality with it because the modular concept of the Quartz Composer can easily broken this way. With that patch, missing functionality can be implemented very often.

Design alternatives

But sometimes there are also other ways to achieve the same functionality without writing a JavaScript by combining other patches. The capability of renaming patches should also be used very often in order to directly clarify the intention of a patch inside a composition. So, for example, a "Conditional" patch that checks if a value is greater zero could be named "postitive number?". With that textual assistance, the developer or others working with the patch would get a better insight into the (maybe foreign) patch. It is a bit like commenting source code. Additional information can also be provided in the comment field below the "Title" field in the Inspector window.

More patches

Other patches that are not explicitly mentioned in this section should individually be explored and the adequacy for a composition has to be judged in the situation at hand.

6.4 Tools running besides the Patch Panel

Additional tools

In parallel to the Patch Panel application, a number of other programs is running. They provide support for rapid prototyping as well, starting from the different proxies running for each iStuff device and ending with the *Event Logger* application that supports the developer in allowing to examine the events posted onto the Event Heap.

Prototyping activities

The different kinds of prototyping activities did not necessarily fit into the scope of the Patch Panel GUI. A cross platform solution for all the proxies is needed since some of them only run on specific operating systems like Linux or Windows due to their vendor support.

Proxy management & event debugging

As the presented Quartz Composer extension is not platform independent, features like proxy management and event debugging were externalized to other applications. that are described in the following.

6.4.1 Proxy Manager

Command line wrapper

As mentioned above, the Proxy Manager application allows to quickly connect iStuff components to an Event Heap in range. Figure 6.3 shows an application screenshot.

Scanning and connectiong

The Proxy Manager automatically scans for Event Heaps available at runtime and displays them on the left side of the application window. Mechanisms to change the Event Heap connections of all displayed proxies are also provided.

Tree view

The available proxies are displayed in a tree view and are sorted by their purpose. For example, Phidgets are displayed in a tab especially reserved for Phidgets. This classification is specified by XML descriptions provided together with each proxy in its directory. The proxies description files are structured as depicted in figure 6.4.

Proxy categorization

Thus, they are categorized as software or hardware proxies. Then it is distinguished by their class (e.g. controllers, Phidgets, etc.). The last distinction is made by their proper names. New categories can also be added at any place in the XML hierarchy. This is suggested by the three-dotted labels in the hierarchy tree in the figure referred to. Additional information like the command line launch command are also provided by the XML description file.

Proxy configuration

Selected proxies are added to the middle pane and can be configured to send a certain event ID with each event or, in case that they register for certain events, to check received events for a specific ID, similar to provider patches inside the Patch Panel GUI. The event ID is specified in the text field of each proxy representation.

Automatic search for proxies

When the application starts, it automatically executes a depth search starting from a specified folder in the configuration file for the application. This search is useful because newly added proxies to a sub-directory are automatically

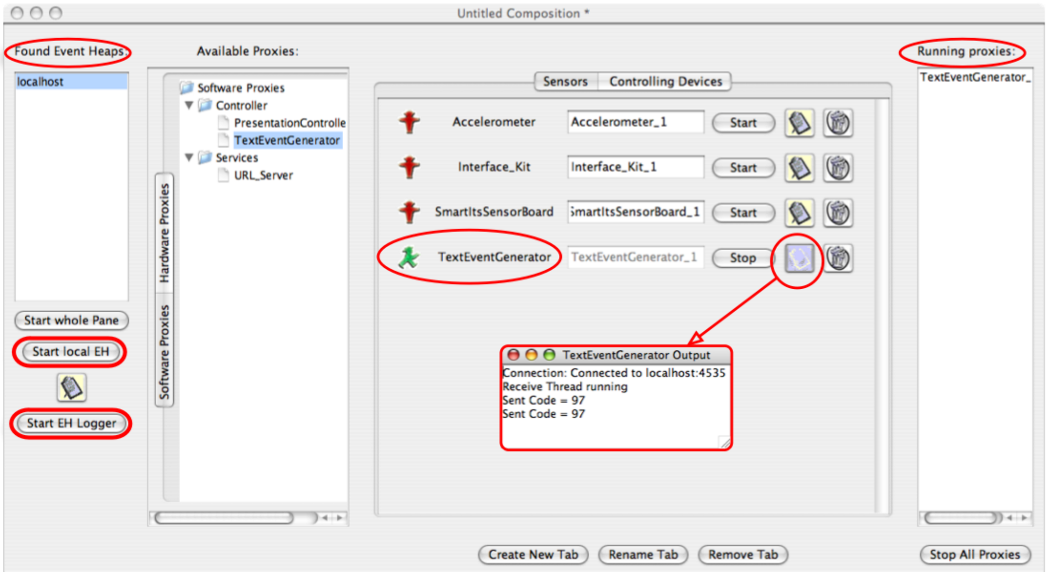


Figure 6.3: A screenshot of the ProxyManager application. Proxies can be arranged on different tabs (middle). On the left the discovered Event Heaps are displayed as well as buttons for launching a local Event Heap and the Event Logger , respectively. On the right side, the currently running proxies are shown. In the displayed situation, a "TextEventGenerator" proxy is running, indicated by the green icon.

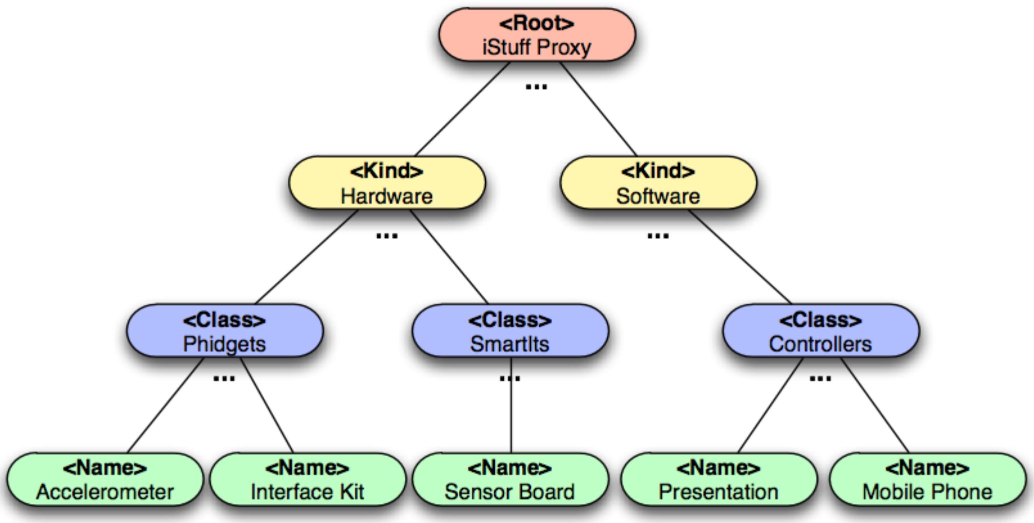


Figure 6.4: A visualization of the XML structure for proxy description files. In the example some proxies were already inserted into the XML tree which is arbitrarily extensible in width.

integrated.

Easy (re)configuration of proxies

The description of the application should emphasize the gain of flexibility and ease of use during the prototyping process since the proxies can directly be launched without the need of navigating through the file system hierarchy via the command line.

Loading and saving

A saving mechanism for setups is also provided such that they can be reconstructed for future usage of the same or similar scenarios. The Event Logger application can also be started directly from this application because in many cases, the Proxy Manager needs to be started before the Event Logger.

6.4.2 Event Logger

Information on posted events

The Event Logger is another Java application that can be run besides the Patch Panel GUI that helps to examine the events that are posted to the Event Heap. Certain events can be filtered out and their fields are shown in detail cf. figure 6.5; all fields inside an event are available inside a detail inspector view(cf. figure 6.6).

Locating errors

With the help of the detailed information, locating errors that are the result of wrongly formatted events is made easier. It can also be checked whether values are really posted as intended or a proxy is working correctly.

6.4.3 Collaboration of the different applications

Interaction example

As a subset of the iStuff components is integrated, a command line wrapper in form of the Proxy Manger is provided and the Event Logger as an analysis tool is presented, a detailed example in form of a narrative scenario should be given to show how the different components can be used in a prototyping context. Figure 6.7 summarizes the described parts once more.

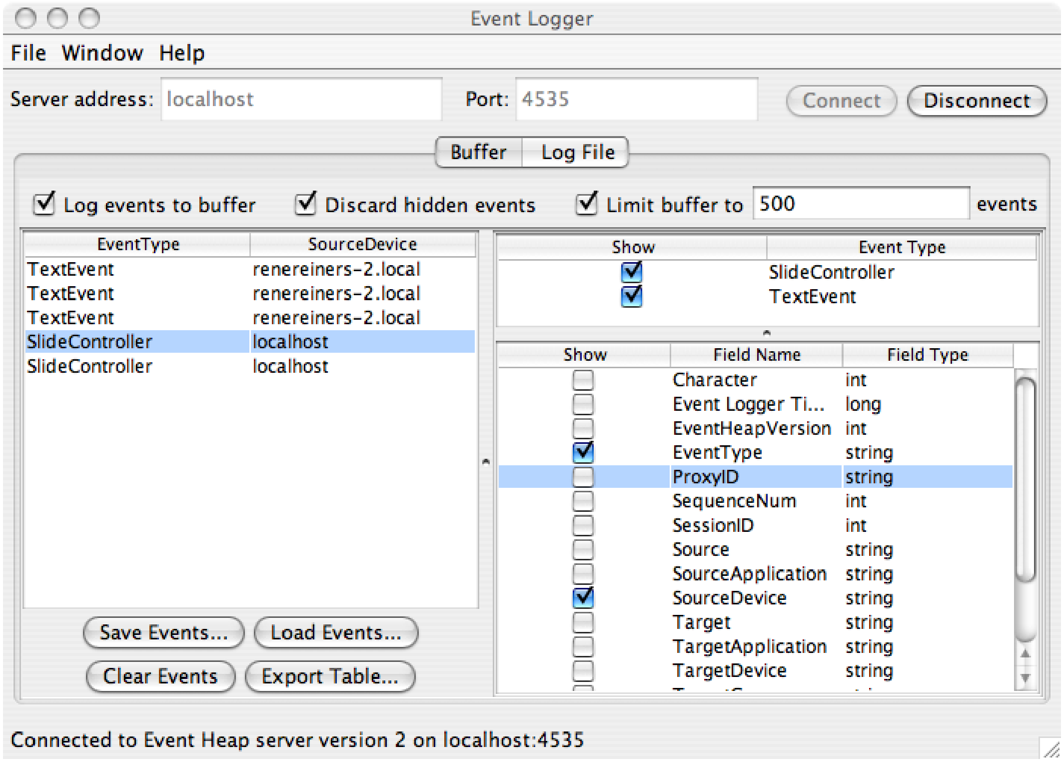


Figure 6.5: The Event Logger application connects to the desired machine an Event Heap is running on and shows either all or selected events that are posted.

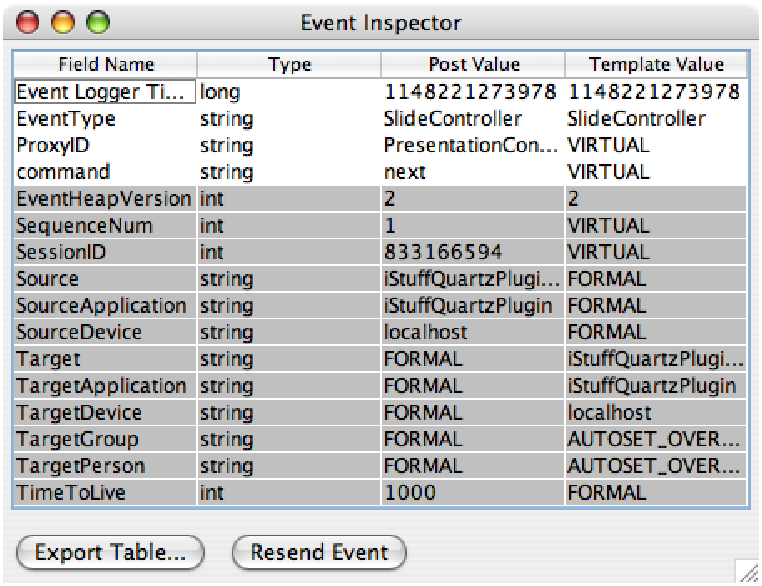


Figure 6.6: The detailed view of an event shows all contained fields.

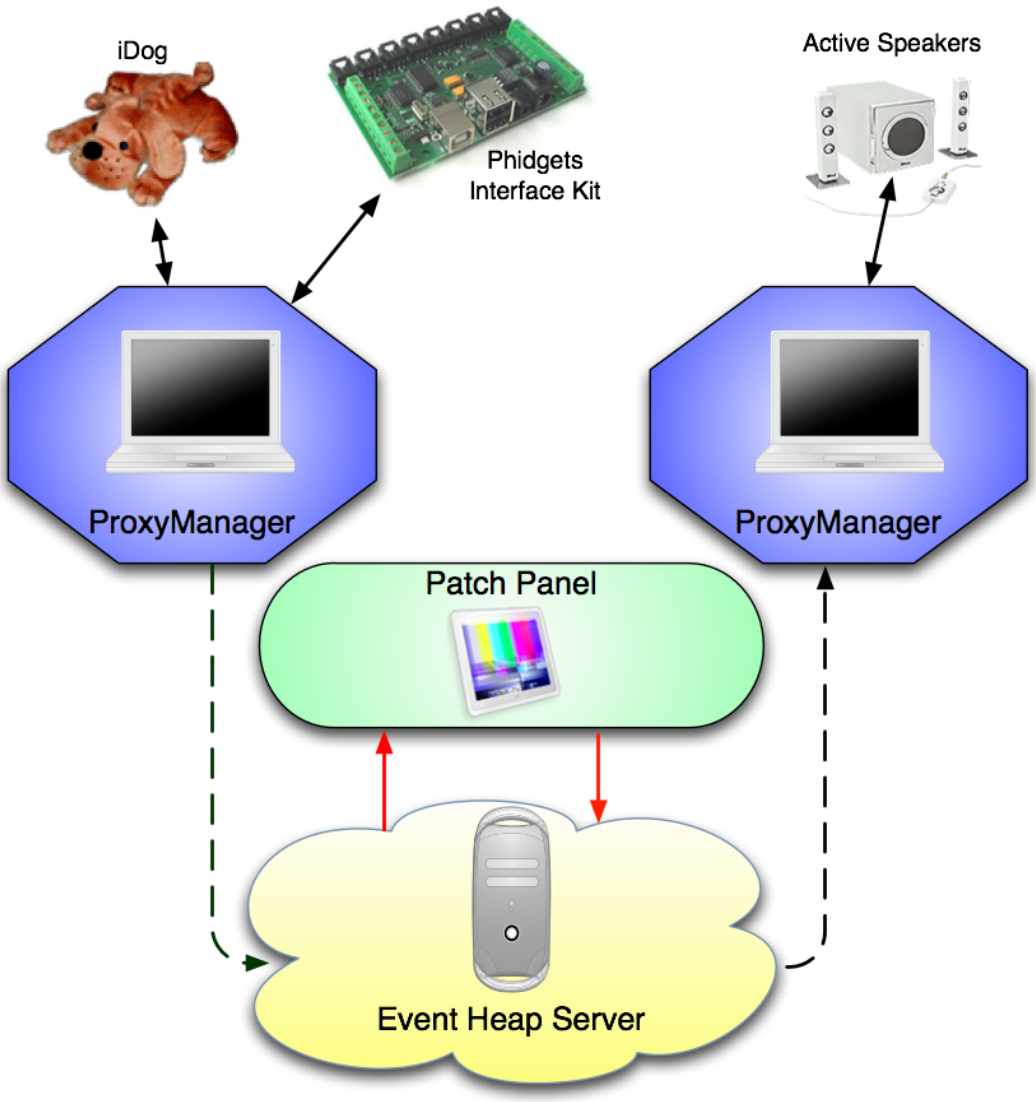


Figure 6.7: On each machine inside the network that should serve as a proxy, the ProxyManager application is run. The Patch Panel is realized by the Quartz Composer iStuff extension.

A multiscreen presentation controlled by a mobile phone

Example scenario

Tom, a 25-year old computer-science student is asked to setup a multi-screen presentation and make use of a new input device from the iStuff toolkit to control the slide transitions.

The same presentation running on two machines

Since Tom is familiar with the iStuff toolkit and its infrastructure, he decides to run the same presentation on two different machines and to connect his mobile phone via Bluetooth to a third machine. The mobile phone support is the latest integration of the iStuff toolkit, coming from the iStuff Mobile project (cf. Memon [2006]). Concerning the presentations, one screen should always show the previous slide of his talk, the other one the current slide he is talking about. The Event Heap necessary as the underlying communication structure is running on another machine inside the room. The new iStuff prototyping suite including the latest version of the Patch Panel GUI is installed as well.

Scenario sketch

Before he starts, Tom sketches the scenario with a software application (since he wants to reuse it for personal purpose later). His setup is shown in figure 6.8.

Presentation controlling proxies

In order to be able to remotely control the presentations, Tom starts up the Proxy Manager on each of the presentation machines and selects the “Presentation Controllers” from the “Software Proxies” tab. The proxies are implemented to listen to events of type “SlideController” but the additional field for the event ID is also required because the proxies should only react on special events of that type.

Proxy launch

For that purpose, Tom sets the ID field in the configuration window is to “PrevSlide” and “CurrentSlide” respectively. Before starting the two proxies, Tom checks whether the correct Event Heap is selected from the list of discovered Event Heaps. Now the proxies that control the presentations are configured and running. Figure 6.9 shows an excerpt of one Proxy Manager application.

Mobile phone connection via bluetooth

As the next step, Tom connects his mobile phone via Bluetooth to his own laptop that is connected to the local network[8]. With the Proxy Manager application he downloaded together with the iStuff package, Tom selects the proxy that posts events from the mobile phone to the Event Heap. The event ID is automatically provided by the application. This time, Tom sees no need to change it.

Mappings

Now that all needed components are connected to the

[8]Details can be found in Memon [2006]

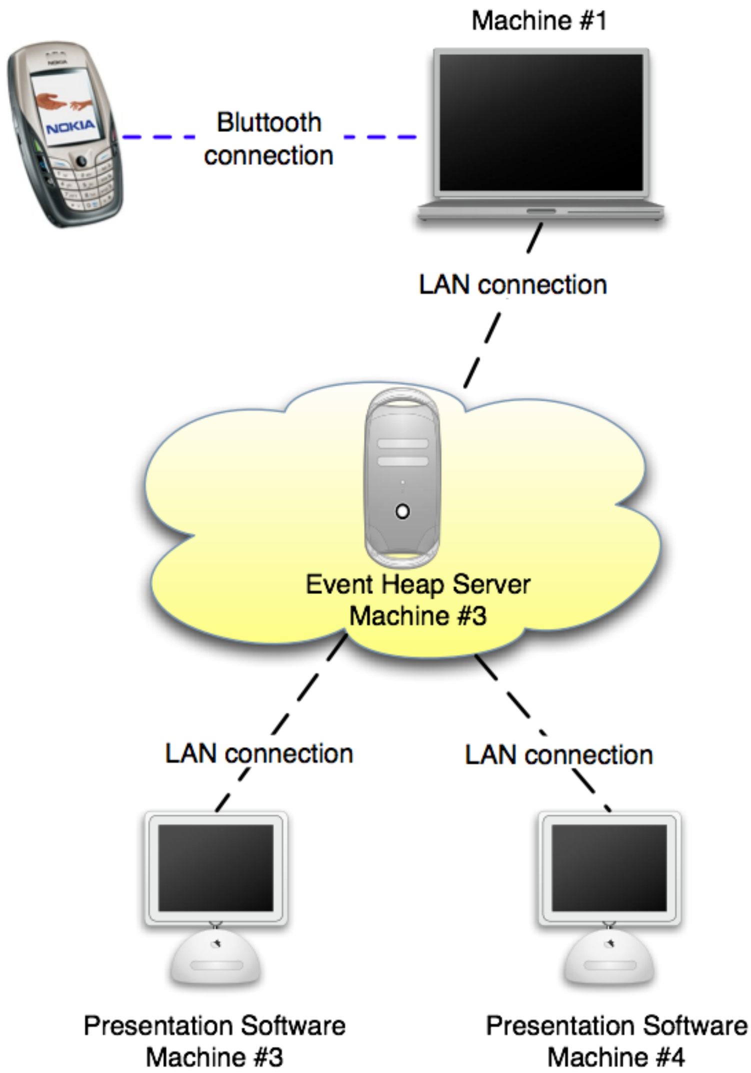


Figure 6.8: A mobile phone is connected via Bluetooth to machine #1 which posts events to the Event Heap Server. The other two machines run the same presentation controlled by the corresponding proxy.

Event Heap and are ready to post and receive events it is time to specify mappings since at the moment, posted events are not meaningful to any component.

"Mobile Phone" patch

On the machine that runs the Event Heap, Tom opens the Quartz Composer extension of the Patch Panel GUI. From the list showing the available patches, he selects the "Mo-

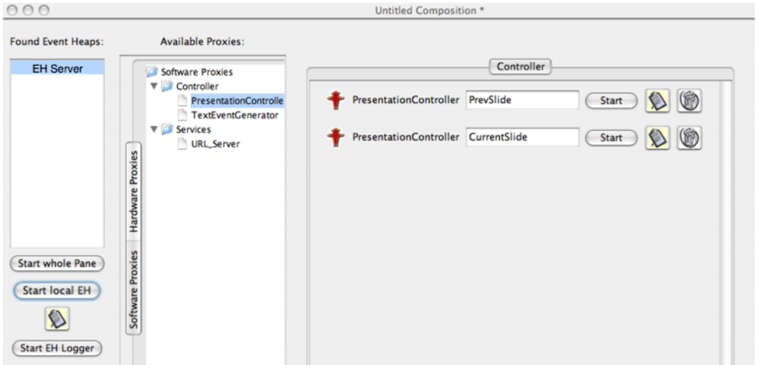


Figure 6.9: A part of the Proxy Manager application for setting up one proxy for controlling the presentation on the local machine.

bilePhone" patch in the "iStuff" category.

Event ID is ignored by default

By default, each iStuff patch tries to connect to the Event Heap of the local machine and then to the first one discovered in case there is no local Event Heap is running. Another default is that provider patches ignore the Event ID until this feature is activated in the "Settings" pane. Tom checks whether the patch connected to the correct Event Heap by using the "Settings" pane of the inspector window.

"PresentationController" patches

Next, Tom places two "Presentation Controller" patches on the workspace. From the purple label and the quality that they only provide inputs that these patches are consumers. That means that they always post their name as the event ID.

Patch renaming

In order to keep the overview, Tom renames them via the inspector to "CurrentSlide" and "PrevSlide", now events with these ID fields are posted to the event Heap. As the proxies on the machines controlling the presentations were configured to listen to events with a specific event ID, they now only react to events posted by their Patch Panel counterpart.

Examine events

After this setup, Tom needs to figure out what ASCII codes are sent by the mobile phone when he presses the keys he wants to use for the presentation. Therefore, he launches the Event Logger application on his laptop and connects to the Event Heap. Whenever he presses a key on the phone, events occur on the Event Heap that contain, among other fields, the ASCII code of that key. They are posted by the proxy he setup some steps ago. Tom decides for the number keys '1' and '3'.

> **DIRECTLY READING OUTPUTS IN QUARTZ COMPOSER:** Another way of accessing the code is to make sure that the "Mobile Phone" patch receives the correct events. When hovering the mouse over the appropriate output port, the received value from the key press is shown.

Specification of a Patch Panel

Finally, everything is set up and running. Events are posted from the mobile phone to the Event Heap. What is left to be done for Tom is to find a valid transformation from the "MobilePhoneKeyListener" patch to the "PresentationController" patches such that they post events for the listening proxies.

Check ASCII codes

He decides to make use of "Conditional" patches that compare values. In this case, the incoming ASCII values are compared to the values Tom discovered with the Event Logger. The comparison is specified with the "Input Paramters" view of the inspector as shown in figure 6.10. He can manually set the ports for the second operand via the same window or by double clicking the second input port since it is not connected.

Final solution

Figure 6.11 shows his final solution. As Tom decided to directly provide a slide number instead of only triggering commands, he also introduced a counting patch that increases or decreases its current value depending on the trigger.

Connecting the presentation controllers

Since both presentation controllers must be triggered, two connections are drawn from the output of the "Counter" patch. The "Math" patch connected between the "Counter" and the "PresentationController" patch for the previous slide is responsible for decreasing the currently counted value by one for the previous slide.

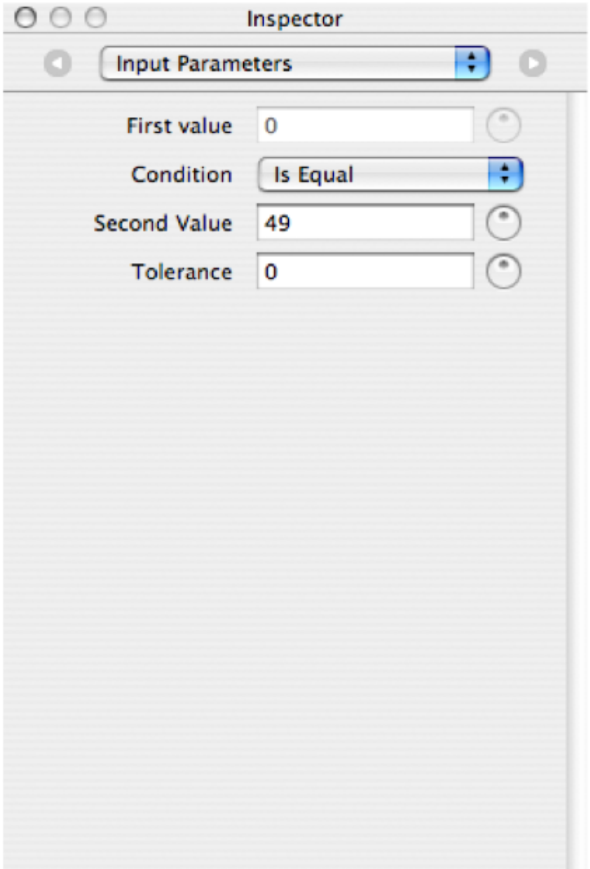


Figure 6.10: The first input of the conditional patch depends on the data coming from the connection and is compared to number 49 with corresponds to the ASCII code for the '1' key.

Different solutions

Of course, Tom might have implemented this example in an easier way by directly connecting the outputs from the conditionals to the appropriate inputs of the slide controllers, namely "Current Slide" and "Prev. Slide". But then, he wouldn't have been able to directly address the slide numbers what could be interesting for future extensions of the composition.

Testing with events

Tom tries out his composition by pressing the defined keys and sees that with each key press, two events (one for each controller) are posted to the Event Heap and that the field for the slide numbers are set correctly. Now he launches the

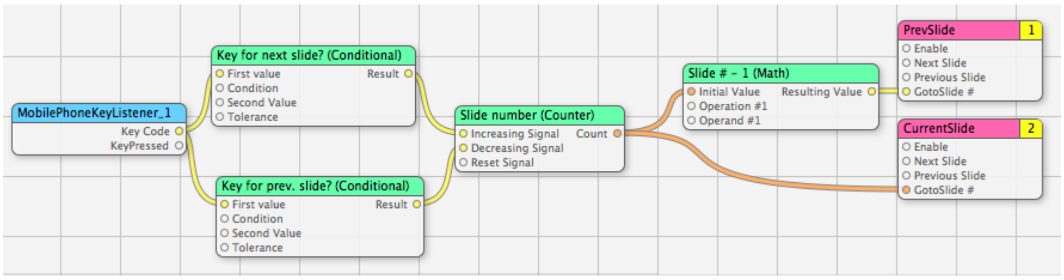


Figure 6.11: One possible solution for the scenario described in section 6.4.3.

presentations on both machines and practices his talk.

Built-in patches

This example already illustrates that the already built-in patches provided by Quartz Composer can also be used for the Patch Panel. Different solutions are always possible to a design problem. With the Patch Panel GUI, they can now quickly be implemented.

Quickly implementation of alternatives

The described alternative could directly be implemented by simply reconnecting the different patches. The controlling keys could also have been changed by changing the conditional.

Reconfiguration at runtime

Once the proxies are configured and running no compilation of the composition or restart of any application is needed. All functionality that is to be tested is specified via the Patch Panel. The only exception raises when proxies should change the event ID they listen to; in that case they have to be restarted which should not be a big issue since the Proxy Manager supports this quick alternation.

EXCHANGING CONTROLLING COMPONENTS:
In the latest generation of Apple Powerbooks a *sudden motion sensor (SMS)* is integrated. Basically, this sensor is an accelerometer that provides three values for possible accelerations in each direction. A proxy was written that is capable of reading out the detected values. With conditional patches, bounds for the values corresponding to the degree of tilting in y-direction can be specified that also act as triggers. The exchange of the input device simply consists in selecting the "PowerbookTiltSensor" patch and attaching its output ports to the existing or new conditionals.
A Phidgets Interface Kit or RFID tag reader could also be taken as input devices; Different kinds of sensors are then attached to the Interface Kit and the mappings are adjusted at runtime. The same principle applies to the tag reader.

6.5 Implemented examples

Only descriptions of mappings

In this section, some implemented examples are presented to provide a feel for the use of the Patch Panel GUI. Since the collaboration of the Proxy Manager, the Event Logger and the Patch Panel GUI were described in the preceding example the following descriptions only refer to the mappings specified inside the Patch Panel GUI. The configuration principle of the proxies posting or registering for events is always the same throughout all the examples.

Rebuilding scenarios from literaure

The scenarios were partially developed for conference submissions (cf. Ballagas et al. [2006a] and Ballagas et al. [2006b] and refer to prototyping scenarios for mobile phone interactions described by Harrison et al. [1998] and Schmidt et al. [1999].

Mobile phone scenarios

Since there was a lot of cooperation with Memon [2006], examples from this field were often taken. Other prototyping scenarios were used in the user study presented in chapter 7 where they are described in detail.

6.5.1 Typing on mobile phones

Proxies for mobile phones

In the scope of the iStuffMobile project (cf. Memon [2006]) a software proxy was created that listens for key presses inside a text window. As soon as character sequences are entered, an event containing the code for each character is sent to the Event Heap. On the Patch Panel side, a "CharacterGenerator" patch listens for events of that type. With the help of the "MobilePhoneController" patch, events that can be interpreted by a software proxy also created with the istuffMobile project are posted to the Event Heap. The software proxy sends - based on the event received - appropriate commands to the mobile phone via a bluetooth connection. So, for example, a type-to-write scenario can be realized when the phone runs a text or short message service application. Figure 6.12 shows the according Patch Panel mapping.

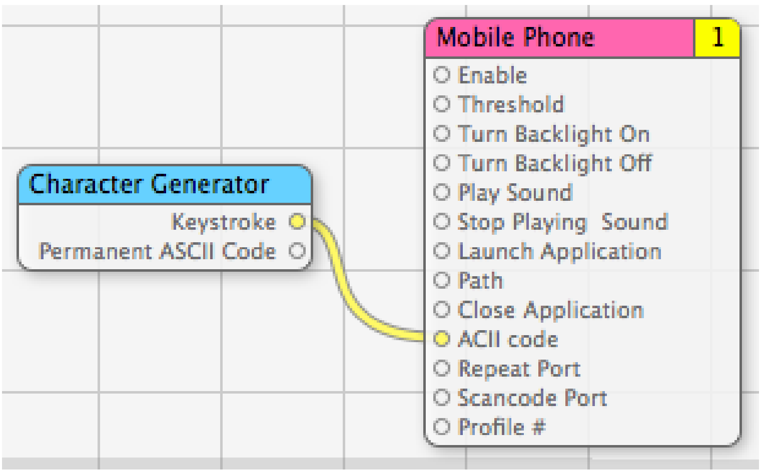


Figure 6.12: A patch that receives events sent from a proxy that receives ASCII codes from the connected keyboard is linked to a consumer patch that sends events to control a mobile phone.

OUTLOOK – SPEECH RECOGNITION ON THE MOBILE PHONE:
This scenario can be augmented with the *iSpeech application*. This dictation software can be trained and ran in the background in such a way that recognized words are directly typed into the textfield. The rest of the composition stays the same. This example also shows how different scenarios can easily implemented by partial changes.

Attached SmarIts

A SmartIts sensor board was attached to a mobile phone. The sensor data is sent to Event Heap via a proxy and processed with a corresponding patch inside the Patch Panel. With the help of Java scripts and conditionals it is possible to implement the *tilt-to-scroll* and the *smart profile changer* scenario described by Harrison et al. [1998] and Schmidt et al. [1999].

6.5.2 Tilt-to-scroll

Menu scrolling

Based on accelerometer data received from a SmartIts Sensor and a touch sensor attached to a mobile phone, menu scrolling should be activated whenever the phone is tilted to a certain degree and squeezed (the touch sensor is pressed). The degree of tilting determines the scrolling speed. The working mapping as well as the accompanying Java script are depicted in figure 6.13. The "Multiplexer" patch holds the ASCII codes corresponding to the scrolling keys of the mobile phone. The appropriate ASCII code is selected depending on the output of the java script. The "Rate" patch controls the sending of a key code to the mobile phone. That way, the scrolling speed can be influenced. The code for the java script is shown in figure 6.14

6.5.3 Smart profile changer

Situation-aware profile changing

SmartIts sensor boards are also capable of detecting the degree of ambient light. With certain thresholds it can be determined whether the phone is inside a suitcase (dark environment) or in a room. If the data from the touch sensor

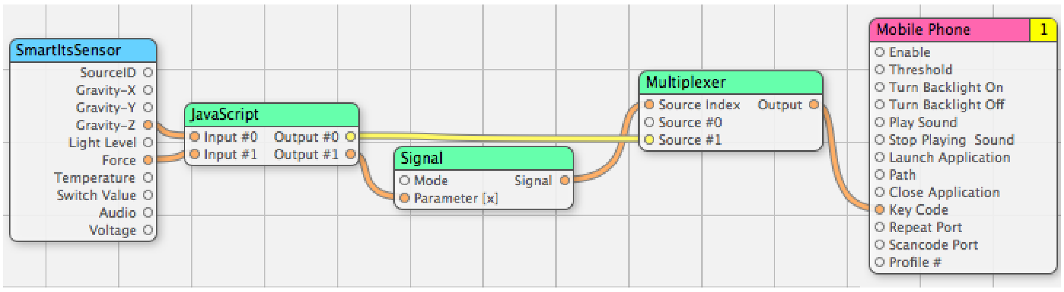


Figure 6.13: The tilt-to-scroll realization with the Patch Panel GUI.

is also taken into account, a change-profile-scenario can be implemented. The assumptions are that in a dark environment the phone should turn off (e.g. at night) and when enough ambient light is detected, it is checked whether the phone is held in hand or untouched (e.g. lying on the table). This test is performed with the help of the touch sensor. When the phone is held, it should only vibrate whereas it should ring when it is not touched.

Numbered profiles

The results of the checks are passed to the consumer patch. Numbers determine what command is sent to the Event Heap and what profile is to be activated in the mobile phone. The different profiles are setup on the mobile phone beforehand.

Alternative: Java script

This scenario could also have been realized with a "JavaScript" patch that yields out a certain number for the "Profile #"input of the "MobilePhoneController" patch but for this example, an alternative is discussed that replaces the JavaScript with several numerical and boolean conditionals.

Alternative composition

Figure 6.15 shows the complete composition that takes light and force values from a SmartIts sensor board. Inside a macro patch whose detailed structure is shown in the lower part of the same figure, it is determined which one of the four possibilities (dark and touched / dark and not touched / light and touched / light and not touched) is true depending on the sensor input.

Calculation of profile number

"Math" patches for each boolean check inside the macro patch calculates the number of the condition (1-4) that is

```
/*
This Script sets the rate for the Signal Patch to fire TRUE from
its output port.
With that TRUE signal, the event rate fired by the Mobile Phone Patch
can be controlled.
The value for the rate (outputs[1]) depens on the rotation in the z-axis.
Inside this script four ranges are defined: Fast/Slow Forward and
Fast/Slow Backward, respectively.

Outputs[0] sends the key code for the mobile phone for scrolling up
or scrolling down - depending on the z-axis, too.

If, additionally, the Force-value is above 10000, the key code for scrolling
is fired. If

If both conditions (Force and Tilt) are FALSE, ASCII code 0 is sent.
*/


// input [1] = Force
// input[0] = Tilt

if (inputs[1]  >= 10000 && inputs[0]  >= 16700000)
{
 outputs[1]=63497;
 outputs[1]=1; // slow rate
}
else if (inputs[1]  >= 10000 && (inputs[0]  >= 16670000 && inputs[0] <
16700000))
{
 outputs[0]=63497;
 outputs[1]=0.5; // fast rate
}
else if (inputs[1]  >= 10000 && (inputs[0]  >= 90000 && inputs[0] <=
200000))
{
 outputs[0]=63498;
 outputs[1]=0.5; // fast rate
}
else if (inputs[1]  >= 10000 && inputs[0]  >=  4000)
{
 outputs[0]=63498;
 outputs[1]=1; // slow rate
}


else
{
          outputs[0]=0;
}
```

Figure 6.14: The Java script to check the degree of tilting and the values sent out by the force sensor on the SmartIts sensor board.

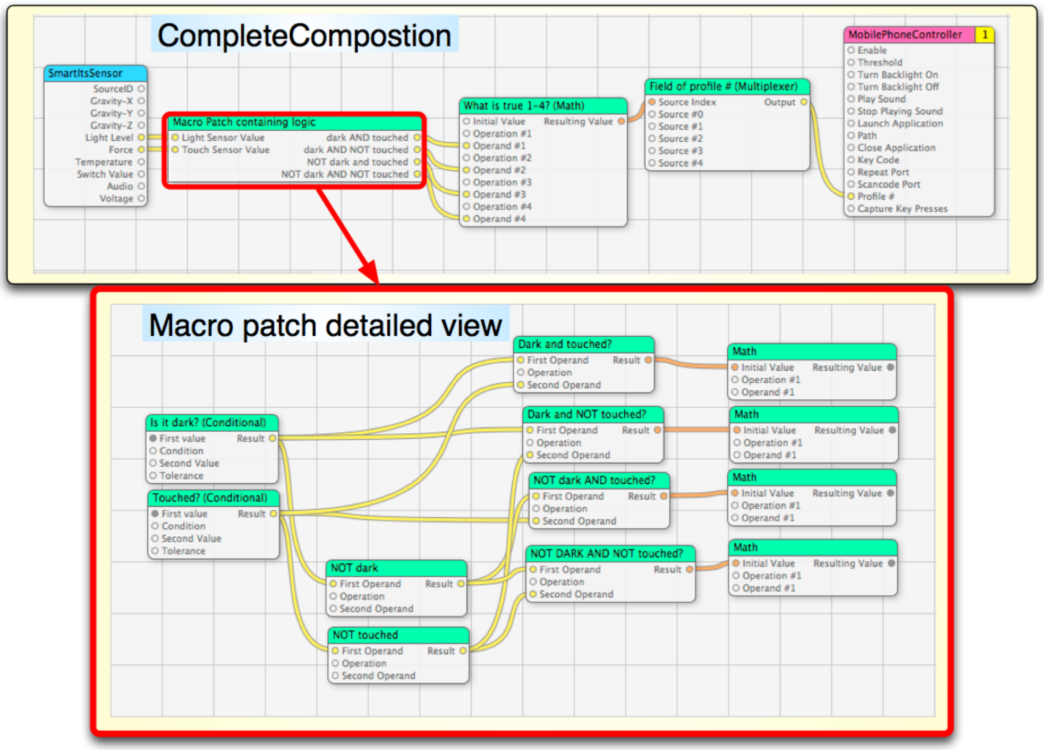


Figure 6.15: Light and force values are read from the SmartIts sensor board. Inside the"JavaScript" patch the thresholds are specified and a profile number is set a t the output. The patch that controls a mobile phone sends and event containing the command for changing to the specified profile.

true. The "Math" patch at the top level of the composition takes the four possible outputs and bundles them to one that is passed to a "Multiplexer" patch. They now have the function of the selector for the preset inputs of the multiplexer. Each set input corresponds to a profile number on the mobile phone. Therefore, the connection to the corresponding input of the "MobilePhoneController" patch can be drawn.

PORTING TO OTHER PLATFORMS:
It is important to see that this scenario can be implemented on any phone for which a proxy can be written. The Event Heap structure and the Patch Panel do not care about hardware specifications, they only pass information, not data. With that concept already stated in the introductory chapters, the iStuff framework and especially the Patch Panel stay device independent.

6.6 Discussion

Extension of initial goals

The initial idea to provide a graphical user interface for the Patch Panel was extended during the development of this thesis. When it was found out that Quartz Composer could arbitrarily be extended in form of custom patches, this new direction was followed. Some examples of prototyping scenarios were shown at the end of this chapter. Making use of the hierarchy mechanism that is provided by macro patches larger scenarios like interactive room scenarios as they are described e.g. by Ballagas et al. [2004] and Humble et al. [2004] are also imaginable.

Interactive room scenarios

By grouping a certain setup in a macro patch representing one specific room scenario, several scenarios in one large composition could be defined. The enabling ports of a macro patch (that has to include a consumer patch) would be published such that the single scenarios could be activated or deactivated at runtime.

Separate applications

During the development phase it also became clear that some features like managing different proxies or the examination of events posted to the Event Heap had to remain external tools. That makes it easier to launch and configure proxies running on different machines. Event checking and debugging can also be performed on different machines and even be distributed to different persons that only examine certain types of events. The division into different applications mirrors the modular concept of the whole iStuff project and leaves open spaces for separate improvements in each field.

To follow: A user study

The development in the scope of this thesis has ended so far and the results have to be evaluated. For this purpose, the following chapter presents a user study in which the participants were given different design tasks. The results should justify the design decision taken during this work.

Chapter 7

Evaluation

Thesis goals were achieved

The development of an extensible graphical user interface for the Patch Panel in the scope of this work has been finished. Additional tools like the Proxy Manger and a couple of custom patches have been added to the whole prototyping suite in order to facilitate the setup and configuration of the iStuff components. The already existing Event Logger has been emphasized as another important prototyping assistance. It was stated that the collaboration of all the presented tools makes the iStuff toolkit powerful in terms of rapid prototyping. The extensibility of each tool as well as the components of the toolkit was shown.

Application of the DIA-cycle

Following the DIA-cycle introduced in chapter 1, the currently presented tools have been developed.

Design iteration

Several design iterations were undergone, starting with a collection of concepts and ideas, going over several iterations and evaluations of paper prototypes together with storyboards. After the concepts and design strategy were cleared, first programming efforts were made.

User study

Now it is time again to make a step towards the analysis part of the DIA-cycle in form of a user study. The user test setup and the execution of the tests are described in detail in the first sections as well as the design tasks the groups were confronted with in order to test the prototyping tools. At the end of this chapter the results of the evaluation are discussed and a conclusion is drawn.

7.1 Preparations for the user evaluation

Hypotheses for the evaluation

The initial aim of this work has been reached: A graphical user interface for the Patch Panel was created that is arbitrarily extensible. The hypotheses made for the user study were derived from comparing the Patch Panel GUI versus the scripting language used before this work started:

1. *The Patch Panel GUI allows faster derivation of prototyping results than the scripting approach*
2. *The Patch Panel GUI encourages more design iterations and refinement of setups during the prototyping process than the scripting approach*
3. *The Patch Panel GUI assists the prototyping process by providing prebuilt atomic functionalities in form of already built-in patches whereas the scripting language does not provide a library of different components with different functionalities.*

7.1.1 Test group

Conceptual understanding of the technique

The initial question: "Who are the users?" was always answered throughout this thesis: Designers who are familiar with the handling of software and have a conceptual understanding of prototyping in ubiquitous environments. They do not necessarily need insight into the hardware details of each part because the iStuff toolkit abstracts from that issue. General logical and numerical concepts, however, should be known such that the transformations between the mappings of events can successfully be specified.

Computer-science students

To suit this user group it was decided to ask 16 graduate computer-science students to take part in the user study. On average, the students were in the eighth semester and most of them attended lectures about the design of interactive systems offered by the department and so were familiar with the concepts of prototyping and iterative design. Since covered partially by the lectures, they were already introduced to the Event Heap communication mechanism. Most of the participants had some basic knowledge about the Quartz Composer application but only two of them judged themselves to be familiar with it.

7.1.2 Setup

Two test runs with eight persons each

The study was performed in two parts with eight participants in each one. Teams consisting of two persons were built. Each team was asked to prototype four design scenarios, each employing different kinds of iStuff components. The general test setup is visualized in figure 7.1 from which it can be seen that each team worked on a single Apple Macintosh G5 workstation.

No communication between teams

Communication between the different teams was not permitted in order to avoid learning effects. At the other end of the room, two more G5 workstations were running that were capable of showing two identical Powerpoint presentations, needed for scenario 1 (cf. section 7.1.3). All machines were interconnected via ethernet, which also meant that the testers needed to pay attention to connect only to their local Event Heap.

7.1.3 Design scenarios

Design challenge scenario

Four design problems each making use of different iStuff components were motivated by a scenario description; the participants should imagine that they were working for a company designing hardware and software applications that process input from devices like sensors or mobile phones and perform certain operations depending on the input.

Exploration of new concept

One day, their advisor enters their office and confronts them with different design problems to which they should

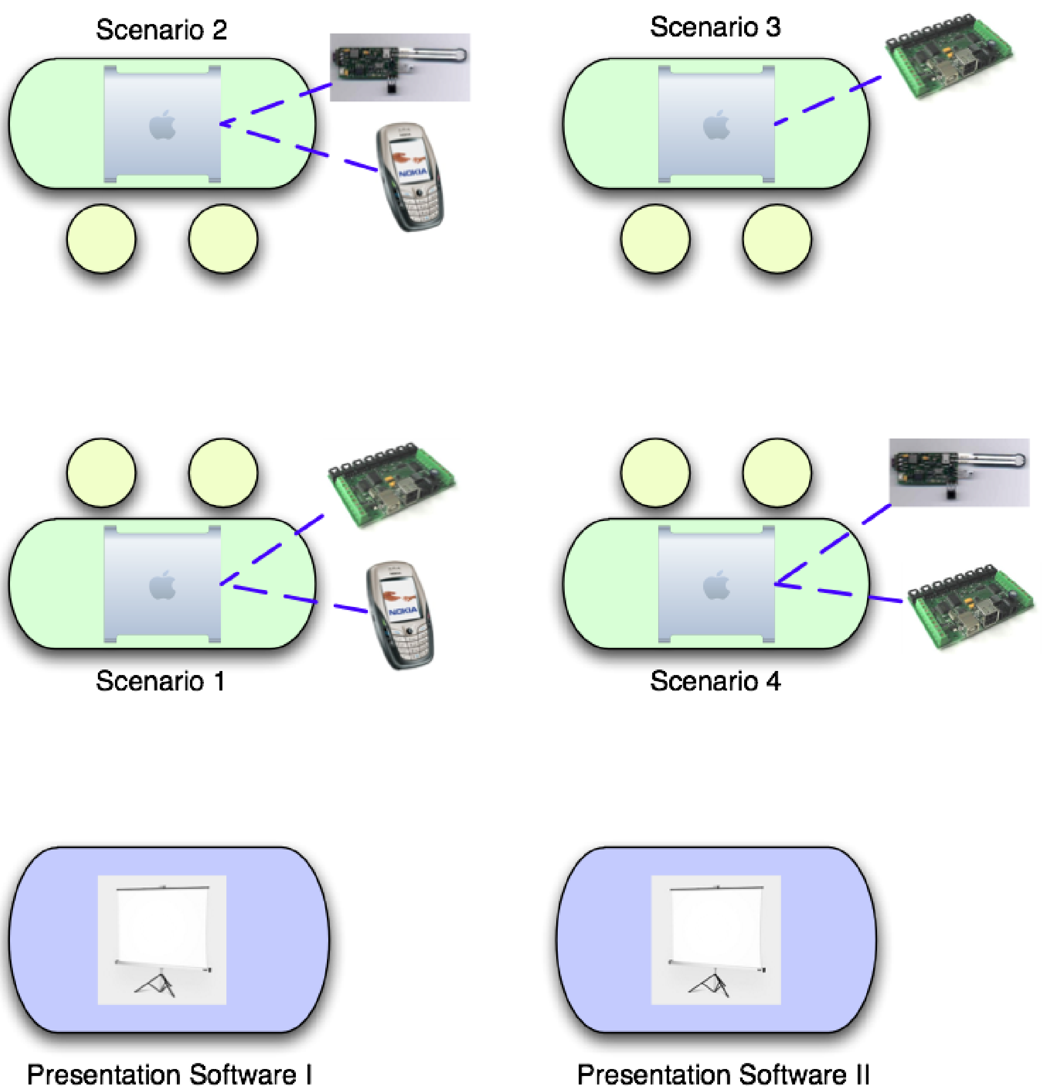


Figure 7.1: The setup for the usertest: Four G5 workstations where one scenario can be prototyped. The participants (yellow circles) are split up into two of two. The same presentation run on the two machines that are separated.

find a solution. It was also to be found out whether the concepts would contain benefits or if they are not realizable. This exploration should be performed with the iStuff prototyping suite available at the company. Appendix B provides the complete scenario descriptions as they were presented to the participants. In the following, the scenarios are shortly described.

Scenario 1: Controlling a multi-screen presentation with a mobile phone

A mobile phone controls two presentations

Like in section 6.4.3, a mobile phone should be used in order to control two presentations running on different machines in the environment. One presentation is showing the current slide whereas the other one displays the preceding one.

Prototyping in a ubicomp environment

This scenario should emphasize the aspect of prototyping in ubiquitous environments in which the components are connected via the network and - in case of the mobile phone - Bluetooth. That means that the interaction takes place in a distributed manner.

Scenario 2: Implement a tilt-to-scroll prototype

Tilt-to-scroll

According to the example presented in section 6.5.2, a mobile phone is equipped with a SmartIts sensor board. A new scrolling mechanism is to be prototyped that activates the scrolling on the mobile phone's menus when a force sensor is pressed and the device is tilted upwards and downwards, respectively. The degree of tilting determines the scrolling speed.

Scenario 3: New concepts for a music player

Prototyping a new music player

A new music player device should be prototyped that introduces new kinds of sensors like touch, rotation, light and force sensors. A software proxy that controls the iTunes application on the local machine was provided in order to mimic the hardware music player. The participants could choose among several Phidgets sensors that were either connected directly via USB or indirectly via the Phidgets Interface Kit.

Scenario 4: Motor control based on sensor data

Automatic balance

This scenario was motivated by a situation in sailing sports where an autonomous motors with a counter weight should keep the boat in balance as soon as the boat tilted above a certain threshold. The motor was simulated by a Phidgets Servo Motor whose controller was directly connected to the local machine via USB. Phidgets sensors as well as a SmartIts sensor board provided the data. The participants should decide what kinds of sensors were best suited to simulate the boat tilting.

7.1.4 Performance

Performance in two runs

This section describes the performance of the user test in details. As already mentioned, the 16 participants were split up to groups of eight that performed the tests separately.

First run

30 minutes introduction

The first group invited attended a 30 minutes introduction on the iStuff project together with the basic functionality of the Proxy Manger application, the Event Logger and the Patch Panel GUI including the original functionality of Quartz Composer.

30 minutes time slots

After the introduction, each team was given a different scenario it should try to implement as far as possible in a 30 minutes time slot. After that time the results were saved at the current states and the groups changed the workstations with their neighbors sitting behind them. Figure 7.2 shows the changing strategy for the complete test scenario.

Introduction of second approach

During another 30 minute time window, the participants were given a short break after which the scripting language for the Patch Panel was introduced and two examples were completed under admission in order to clarify the usage.

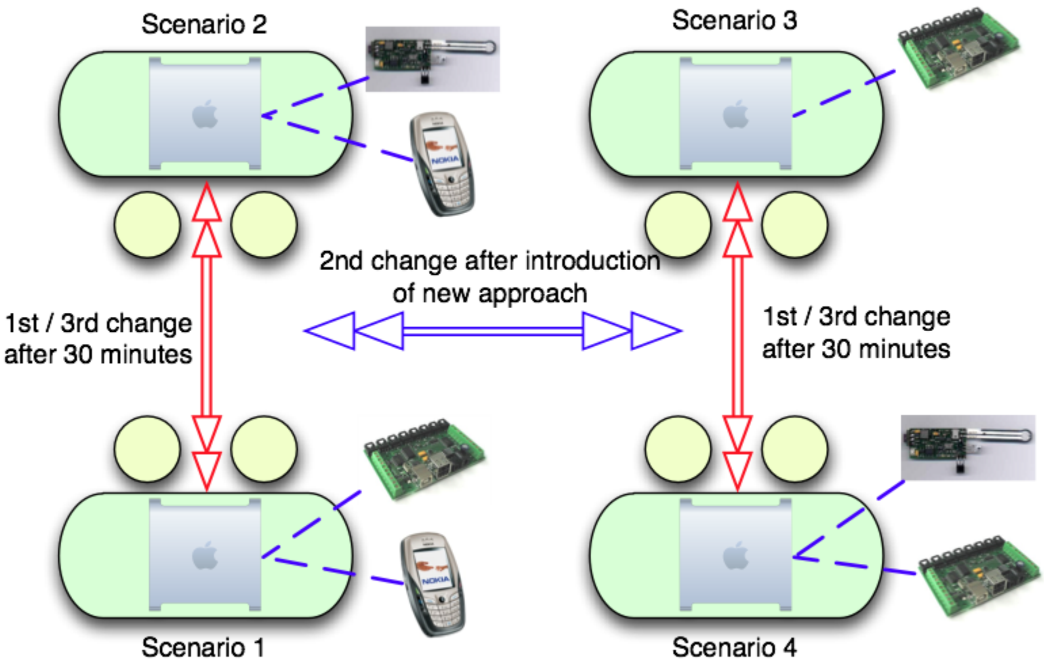


Figure 7.2: After the first 30 minutes, the workstations of the teams sitting back to back were changed. Then, after the introduction of the alternative concept, the workstations not yet used were taken. The last change was performed after the third 30 minutes time slot.

Two scripting language scenarios

The participants were given a second introduction after which they were asked to change to one scenario they haven't worked on so far. Again, after 30 minutes the teams should try to complete the fourth scenario they had not touched so far with the same amount of time.

Second run

Cancel out learning effects

The second run of the user test introduced the eight new participants to the iStuff project but presented the Patch Panel implementations in reversed order such that evaluation results are independent from the order the different approaches were presented. This change in ordering was especially designed to cancel out learning effects.

Scripting language first

Thus, first the scripting language was presented, two scenarios should be prototyped and afterwards the Patch

Panel GUI was explained. The remaining two scenarios were then prototyped with the latter. Table 7.1 shows the setup in form of a matrix.

Test Run #1				
Group	A	B	C	D
Patch Panel GUI	Scenario 1	Scenario 3	Scenario 2	Scenario 4
Patch Panel GUI	Scenario 2	Scenario 4	Scenario 1	Scenario 3
Scripting Language	Scenario 3	Scenario 1	Scenario 4	Scenario 2
Scriptiing Language	Scenario 4	Scenario 2	Scenario 3	Scenario 1
Test Run # 2				
Group	E	F	G	H
Scripting Language	Scenario 1	Scenario 3	Scenario 2	Scenario 4
Scripting Language	Scenario 2	Scenario 4	Scenario 1	Scenario 3
Patch Panel GUI	Scenario 3	Scenario 1	Scenario 4	Scenario 2
Patch Panel GUI	Scenario 4	Scenario 2	Scenario 3	Scenario 1

Table 7.1: User test scenario completion matrix.

Asking for feedback

Demand for criticism

During the whole evaluation it was made clear that it was not the users who were tested but the software they were working with. With these statements, an agreeable test environment should be created and criticism on the software should be encouraged in order to get a lot of feedback also from comments.

Post-evaluation questionnaire

After each test run, the participants filled out a questionnaire that asked about general impressions and the preferences for one of the Patch Panel versions. Their efforts were rewarded with packets consisting of giveaways like pens and t-shirts.

7.2 Evaluation results

Time to complete first version

During the evaluation, the time until the subjects had a basic (but maybe still faulty) prototype was taken as well as the number of iterations they performed in order to improve the scenario. An iteration was defined as every change of the prototype from the first basic one. After the test scenarios, the participants filled out a questionnaire which can be found in appendix C that mostly presented Likert-scales about what approach the subjects felt more comfortable with and which one seemed more promising for the future. The questionnaire evaluation should only give suggestions and expose tendencies.

Freedom in design

The Patch Panel GUI allows a certain degree of freedom in terms of building a working solution. As appendix D discusses in more detail, compositions may vary in their structure i.e. users can derive different solutions by choosing different patches to complete their tasks. This freedom in design also shows that the Patch Panel GUI represents a flexible tool to support rapid prototyping where different solutions can be built and evaluated.

7.2.1 General results

Patch Panel GUI preference

To present the general result first: More than 90% of the participants preferred the graphical approach but also provided some critical thoughts that are presented in section 7.2.3. They were given ten questions that should be judged by discretely scaled answers that were scored with five as the best rating and one for the worst one. The remaining questions should encourage feedback to different aspects of the Patch Panel GUI.

Potential in the Patch Panel GUI concept

The questions asking for the development capabilities with the Patch Panel GUI compared to the scripting approach and whether the users were able to imagine more scenarios where ubiquitous devices could be configured like in the presented way were answered with an average score of 4.5 and 4.2 out of 5, respectively. This suggests the continuation of the Patch Panel GUI development in future.

Understandable metaphor

The data flow metaphor usefulness was scored with an average of 3.9 out of 5 which justifies the applicability of this metaphor for the event passing concept of the iStuff toolkit although the concepts should be explained in a longer session than done for the user study.

Built-in QC patches were used

An average score of 3.2 out of 5 for the question if the already built-in Quartz Composer patches were used a lot supports the decision to extend Quartz Composer and therefore benefit from already included functionality.

Understandable introduction

Four points for the question whether the concepts of the iStuff project were understandable shows that the proven concept of the iStuff project was well explained in the scope of the user test and the participants did not struggle with the overall understanding.

Future extensibility

The graphical approach in form of the Patch Panel GUI was felt as being extensible in the future which is proven by a score of 4.4. This result encourages the extension of the iStuff prototyping suite and strengthens the decision that was made to develop a software framework that makes the integration of new components very easy (cf. section 6.1).

Need for graphical support

With a score of 4.6, the graphical approach was preferred over the scripting language that achieved an average score of 1.6. These results definitely show the need for graphical support for the Patch Panel and moreover that the current implementation can successfully be applied.

Powerful FSM support

The scripting approach was felt as being more powerful with an average score of 3.1. This justifies the need for state machine support in future versions of the Patch Panel GUI as well as several other custom patches that support the prototyping process. The inner structure of the passed events should also be conveyed in a better way. In following iterations of the Patch Panel GUI existing features will be improved and new ones added.

More design iterations

More than 90% expressed to be more encouraged to undergo more iterations with their design using the graphical approach. It seems from this result that the general willingness to refine designs is shown.

7.2.2 Statistics

Measurements

In this section, the statistic results of the user test should be presented. In table 7.2 the completion times for a first prototype and the number of iterations counting since this version are summarized. Almost every group managed to get to a first prototype with the Patch Panel GUI whereas the scripting approach often was too hard for them, probably because the participants were confronted with it for the very first time. Unfortunately, in the tilt-to-scroll scenario only one group managed to implement a first prototype with the GUI but others were on the right track, too.

Run #1	**Scenario 1**	**Scenario 2**	**Scenario 3**	**Scenario 4**
PP GUI				
First iteration	15 min.	13 min.	15 min.	30 min.
Number of iterations	5	4	3	1
change of workstations				
First iteration	20 min.	-	12 min.	19 min.
Number of iterations	5	-	3	1
PP Script				
First iteration	-	-	22 min.	-
Number of iterations	-	-	2	-
change of workstations				
First iteration -	20 min.	-	-	
Number of iterations	2	-	-	-
Run #2	**Scenario 1**	**Scenario 2**	**Scenario 3**	**Scenario 4**
PP Script				
First iteration	-	-	25 min.	-
Number of iterations	-	-	2	-
change of workstations				
First iteration	15 min.	-	-	20 min.
Number of iterations	5	-	-	4
PP GUI				
First iteration	19 min.	-	13 min.	11 min.
Number of iterations	4	-	2	2
change of workstations				
First iteration	10 min.	-	30 min.	14 min.
Number of iterations	5	-	2	3

Table 7.2: Statistic results of the user tests.

Statistical significance

T-test

The measured results were checked for statistical significance with a t-test for unpaired groups. Since there were concrete measurements for time and number of iterations as well as two conditions, namely the Patch Panel GUI versus the Patch Panel scripting language, the test method is appropriate. Whenever no solution for a scenario could be found, a value of 31 minutes was chosen as indicator that the time was exceeded. Figures 7.3 and 7.4, respectively, show box plots of the collected results.

Box plots

The box plots show that times needed for the completion of a first prototype were always shorter when using the Patch Panel GUI. Also, the number of design iterations lay above the results of the Patch Panel scripting language.

Combining scenarios

To show the statistical significance of the results, all scenarios were combined and the overall prototyping times as well as the total numbers of iterations for each approach (Patch Panel GUI vs. Patch Panel Script) were compared. A box plot of the comparison is shown in figure 7.5.

P-thresholds

Since the results are similar to a gaussian distribution, a t-test for unpaired groups was applied. The p-threshold to indicate the statistical significance was set to $(p < 0.01)$. The t-test yielded p-thresholds of 0.1% for the time measurement and 0.7% for the number of iterations which proves the statistical significance of the evaluation results and justifies the drawn conclusions.

Drawn results

Useful GUI support

It can be seen from the results is that for users who are confronted with the iStuff toolkit for the very first time, the general concept was understandable. The prototyping process, however, could only really be supported with the graphical user interface. The learning rate with the Quartz Composer modification was much higher whereas the scripting language requires a long introductory phase. Thus, in terms of enabling developers to *quickly* set up a

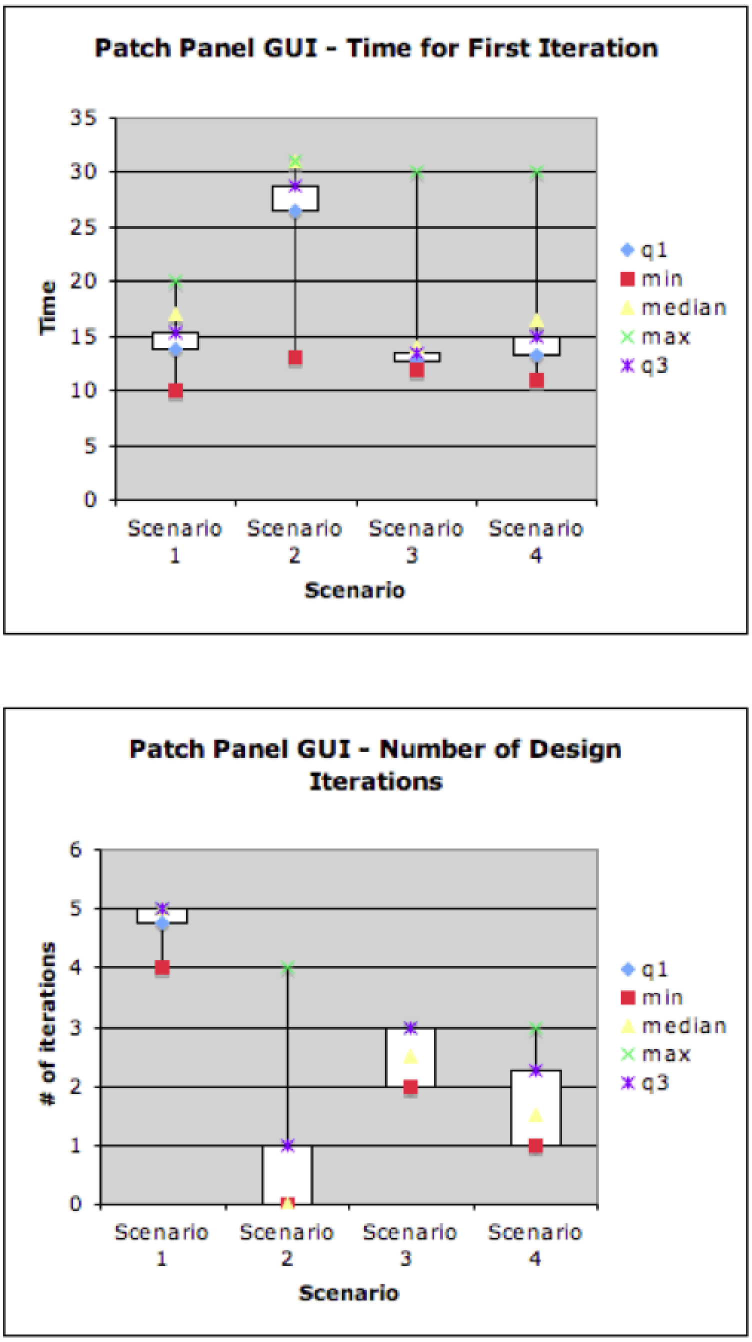


Figure 7.3: Box plots showing the results for the Patch Panel GUI.

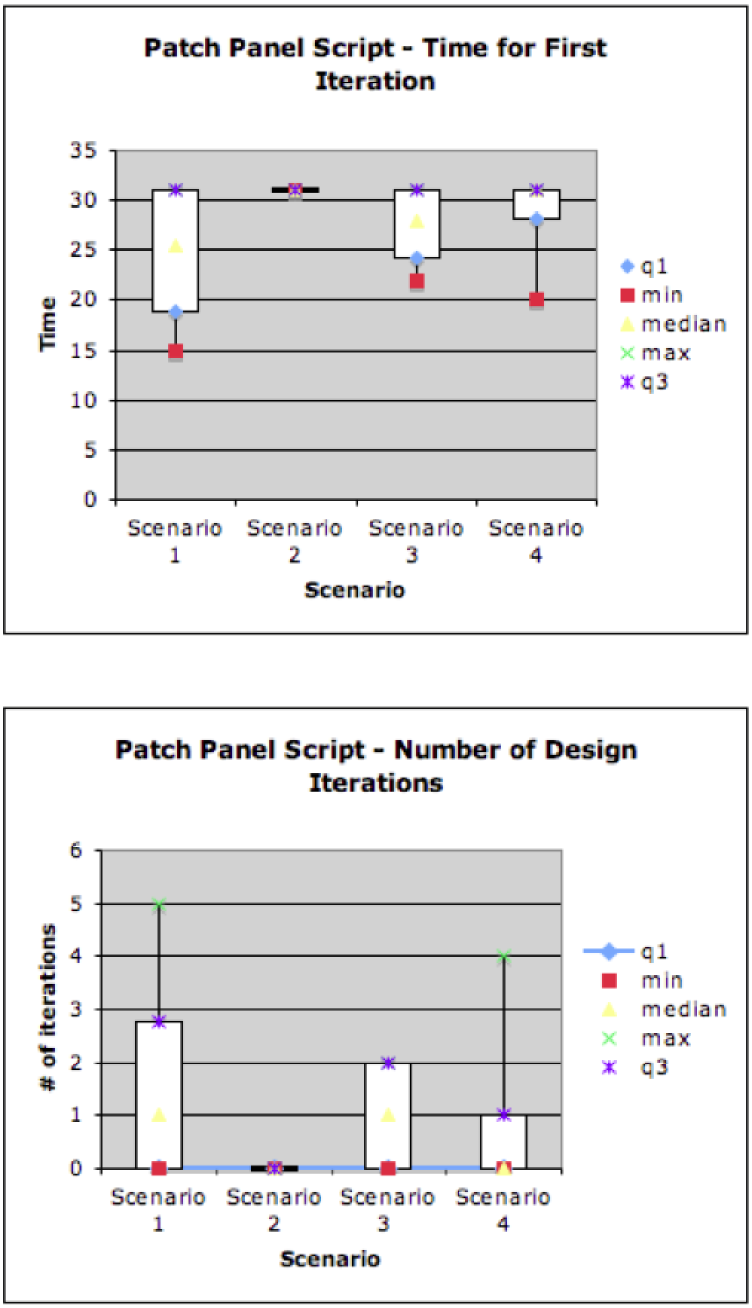
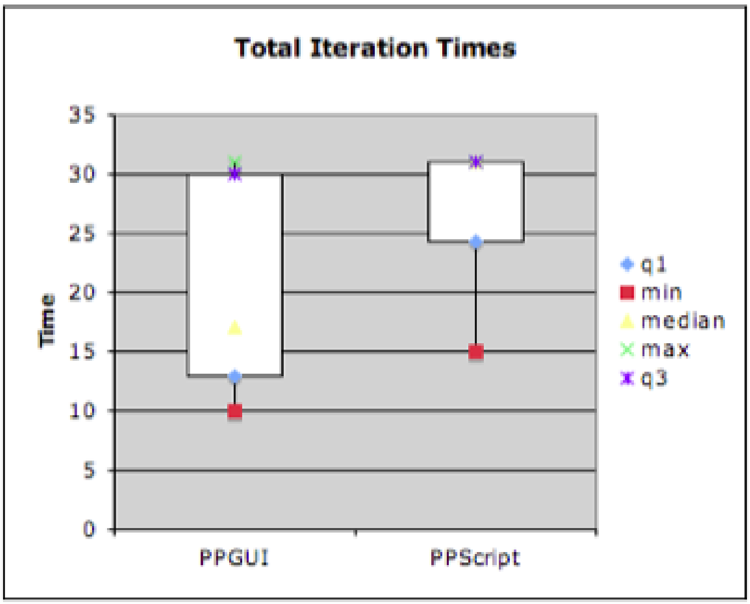


Figure 7.4: Box plots showing the results for the Patch Panel scripting language.



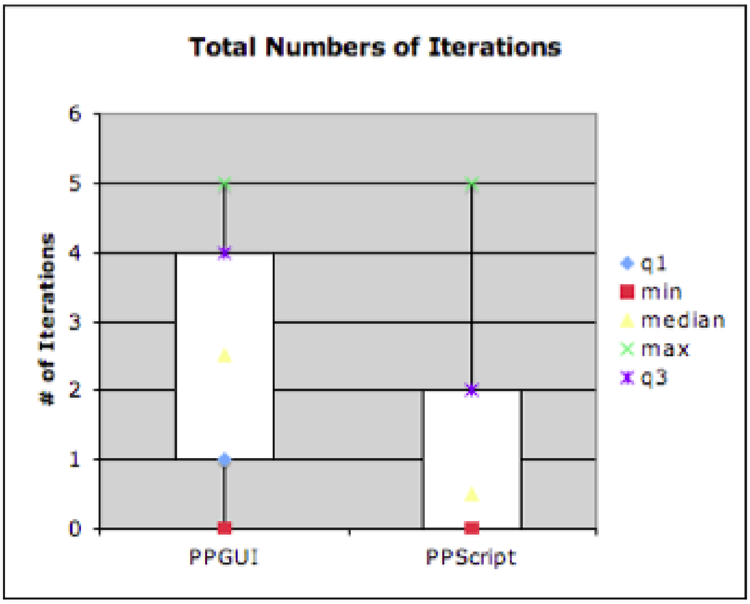


Figure 7.5: The results of the four scenarios were combined and the results plotted for each approach.

scenario, the Patch Panel GUI is the tool of choice.

Better refinements with GUI

Concerning the number of iterations, it can be seen that in all cases the number of iterations is higher with the Patch Panel GUI which is a benefit for the design; with each iteration, improvements are incorporated. One might also argue that this number would have increased if the time window had been larger. In some situations the participants stated that they did not want to refine their designs any further.

7.2.3 Additional feedback

Additional comments

The participants were also given a chance to freely express their criticism of the system, tell what features or patches were missing or what should be changed in future version.

Scripting approach was felt as being immature

It came out, that the scripting approach was judged as immature although it has been used for a long time inside the iStuff project. This point was stated because the users had to apply a lot of workarounds to achieve a working solution. The results demonstrates the difficulty in creating a custom scripting language. Although it was refined and tested for over two years, the testers were still able to find new bugs.

Demand for more incorporation

Another desired feature was that the different applications should be presented in a more condensed or grouped way. The problem with this is that not everything should be integrated into one large application (cf. section 4 for a detailed explanation) because the problem addressing the configuration of proxies should be part of a separate application like the Proxy Manager. The Patch Panel is not part of that. Maybe the Event Logger application could be melted with the Proxy Manager such that events sent by the configured proxies could directly be analyzed.

Ideas for new patches

A new patch like a "Toggle Switch" was demanded that alternatively puts out a value depending on the input and alternates between them with each trigger. This demand demonstrates a need for state machine support, too.

Strings on trigger

Another desired patch should send out a string whenever a certain integer or boolean input was received.

Recreation within Quartz Composer

Although this can be created with the help of different patches in Quartz Composer maybe a patch that directly implements that issue would increase the productivity. It is to be analyzed how often such a design situation will occur in the future.

More sensors

It was also asked for the integration of more sensor types. That should not be a problem for future versions as pointed out in section 6.1 where the extensible framework is discussed.

7.3 Summary of the results

Successful Patch Panel GUI integration

The user evaluation justified the introduction of a graphical user interface for the Patch Panel as it encourages designers to try out several design strategies and refine them in more design iterations. From the statistical results presented in section 7.2.2 it can be derived that the second hypothesis (cf. section 7.1) is proven. The iStuff concept is much easier to apply with the graphical assistance given by the Patch Panel GUI data-flow metaphor.

Quick results

For the users, a composition was easy to create and understand. After a much shorter orientation time, first results with the GUI were gained. It can be argued that with more design time, the participants would have been able to create more precise and elaborated designs than they already had. However, the concept of the graphical approach was widely accepted by over 90% of the participants and preferred over the original scripting approach. With this result, the first hypothesis (cf. section 7.1) is proven, too.

Reusable components

Looking at the evaluation of the user feedback, it can be seen that the participants also used built-in patches that were already provided by the original Quartz Composer application in order to derive solutions. The built-in functionalities offered assistance to the design tasks in contrast to the scripting language that does not offer reusable components. These results justify the third hypothesis (cf. section 7.1).

Integration of FSM support

When the capabilities of the scripting language compared to those of the Patch Panel GUI were discussed the desire for state machine support for the GUI became aloud. As already outlined during this work, state machine support is definitely scheduled for future work. The user feedback strengthens this decision.

Feedback collected for future work

The Patch Panel GUI was realized together with other prototyping assistances and widely accepted by users. The feedback has to be incorporated into ideas for future work. Some of them are described in more detail in the next chapter.

Chapter 8

Summary and future work

"It [The Patch Panel GUI] seems to be a very promising approach. Keep it up!"

—Comment from one of the user test questionnaires.

Achieved goals and scheduled projects

The last chapter of this work summarizes the goals of the thesis and presents the derived solutions. The last section describes concepts and ideas that could not be realized in the available time. Therefore an outlook and suggestions for future work and extensions of the iStuff project, especially the graphical support for rapid prototyping, are given.

8.1 Summary and contributions

Provision of graphical support for the Patch Panel

The aim of this work was to augment the existing iStuff prototyping suite with a graphical user interface that supported the specification of event mappings inside the Patch Panel intermediary service. Before this work was started, these specifications could only be configured with a hard to learn use scripting language. A basic GUI allowed the formulation of very simple, predefined mappings and the import of working scripts. This concept did not suit well into the iStuff concept as the iStuff project wants to support the rapid prototyping process in ubiquitous environments. Concerning the distributed software architecture as well as the way hardware components are integrated into the iStuff framework, this goal has been accomplished. However, a way to easily configure the components and quickly change the interaction between the components was still missing.

Stable foundation for future extensions

With the Patch Panel GUI, the Proxy Manager application and the Event Logger, capabilities for enabling rapid prototyping of new compositions and interaction techniques are provided. The prototyping suite developed in this thesis represents a stable foundation for future extensions.

Live modification of the setup

The Patch Panel GUI, implemented as an extension of the Apple Quartz Composer application, allows the specification and modification of new and existing mappings at runtime without the need of recompiling or restarting the setup. Even the connection management can be altered without requiring any application to be restarted. The choice of Quartz Composer was made on the one hand because it already provided much of the desired functionality for the Patch Panel GUI in terms of graphical representation issues. On the other hand it gains more and more popularity inside the design community and other groups also started to develop their own extensions - although those are centered around different topics. But this trend indicates that the Quartz Composer application will remain under constant development and improvement. This can only be beneficial for future versions of the Patch Panel GUI.

Mimic existing scenarios

The iStuff developed prototyping suit is flexible enough not only to easily integrate new hardware parts but also to recreate scenarios provided by other research groups. Although completely different hardware and design concepts were used, similar components of the iStuff toolkit can be combined in such a way other implementations can be mimicked without having to know details about the original hard- and software components. The tilt-to-scroll-scenario as well as the profile-change-scenario developed for mobile phone applications are two examples for this (cf. sections 6.5.2 and 6.5.3).

In order to derive the solution presented a survey on related work was performed in order to gather useful concepts of graphical representations of mappings and interaction techniques implemented in other applications.

A survey on related work was performed

From that survey, rough design patterns could be extracted and incorporated into a feature list from which early paper prototypes were developed and evaluated. Chosen storyboards showing possible paths of interaction were presented in this work. From the whole set of paper prototypes, a virtual image of the application that was to be developed could be created. After several evaluations of the paper prototypes the resulting conceptual user interface strongly reminded of the Quartz Composer application.

Rough design patterns were extracted

As a consequence, Quartz Composer was examined in detail and ways to write custom extensions for the application were found. After much work that tried to find correct ways of integrating new components (since Quartz Composer is completely undocumented), a subset of iStuff components could be integrated in order to show that Quartz Composer is well suited as a graphical user interface for the Patch Panel.

Detailed examination of Quartz Composer

After first successes, a todo-list was created that summarized the features that still had to be integrated. The result consisted of the Quartz Composer extension as the Patch Panel GUI together with custom patches that help in the prototyping process. The Proxy Manager application allows fast configuration of different proxies. The use of the Event Logger provided by the iROS package give users a chance to examine the events posted to the Event Heap.

Creation of a feature list

The work was evaluated by a user test session in which several design tasks had to be accomplished. The evaluation setup allowed a comparison of the original scripting language and the newly created graphical support for the Patch Panel. Its results were justified with a t-test that ensured the statistical significance of the results. The strong preferences lay at the Patch Panel GUI and more tasks could be accomplished with it. The graphical approach also encouraged more design iterations.

Evaluation with user tests

Feedback from the user tests was collected with question-

User feedback for future work

naires where one part of the questions should be answered on a Likert-scale and the other part gave the opportunity of expressing custom ideas and criticism. They are presented in this work as well as the statistical results. The comments and ideas were also incorporated into future work additionally to the ideas that came up during the development process of this work.

8.2 Future work

Time restriction

Some tasks that would also have fit into the scope of this thesis had to be left out because the development time was consumed and therefore some parts could not be completed. Thus, instead of hacking together a lot of new features that might work, it was decided to leave them open for future work and to implement current features in a stable and reliable way. Since a flexible software framework is provided by this work, additional ideas should be implementable in a fast way.

Planned FSM support

Features that are thought of at the moment are ways to integrate state machine development in form of new patches into Quartz Composer. They play an important role in the scripting approach to configure the Patch Panel (cf. Ballagas et al. [2004]). There are applications where state machines become a very useful tool to specify certain functionalities that require state transitions. This question should be explored in more detail in the future.

Automatic script import

Another idea is to automatically process scripts written in the scripting language for the Patch Panel (cf. Yu [2006]). It does not seem realistic to expect the built of a completely functional Quartz Composer composition but maybe parts of the scripts could be processed in such a way that state machine patches are configured by them.

Visualization aids

The plotter patch as well as other visualization assistance have been scheduled. While this work is reviewed, these patches are already under development but unfortunately could not be completed until the due date of the thesis.

Event Logger integration

Mechanism that integrate the Event Logger into the Proxy Manager or also into the Patch Panel GUI are considered at the moment. The standalone version, however, should also be kept for independent and distributed event debugging.

Composite patches

The evaluation turned out some new ideas for patches that could on the one hand be rebuilt with original Quartz Composer components but, on the other hand, might be a valuable help for the prototyping process by summarizing often used functionalities in one module.

Easy integration of new components

The software framework provided with this thesis facilitates the future integration of existing and upcoming iStuff components. Like the proxy strategy applied for the Event Heap connection management, the framework supports the easy incorporation of new components. As an example, new sensor devices or software proxies could be named.

Scheduled features

Additional features like those described in chapters 4 and 5 are encouraged to be implemented in the future. Hopefully, Apple will give developers more insight into the Quartz Composer framework such that custom extension become more powerful in the future.

Foundation for future development

Finally, it is to be stated that the iStuff project seems to be on the right track for supporting the rapid prototyping process in ubiquitous environments. The underlying architecture, the integration concept for new components and the newly added graphical support should provide a good foundation for the future development.

Appendix A

Storyboards and paper prototypes for the Patch Panel GUI

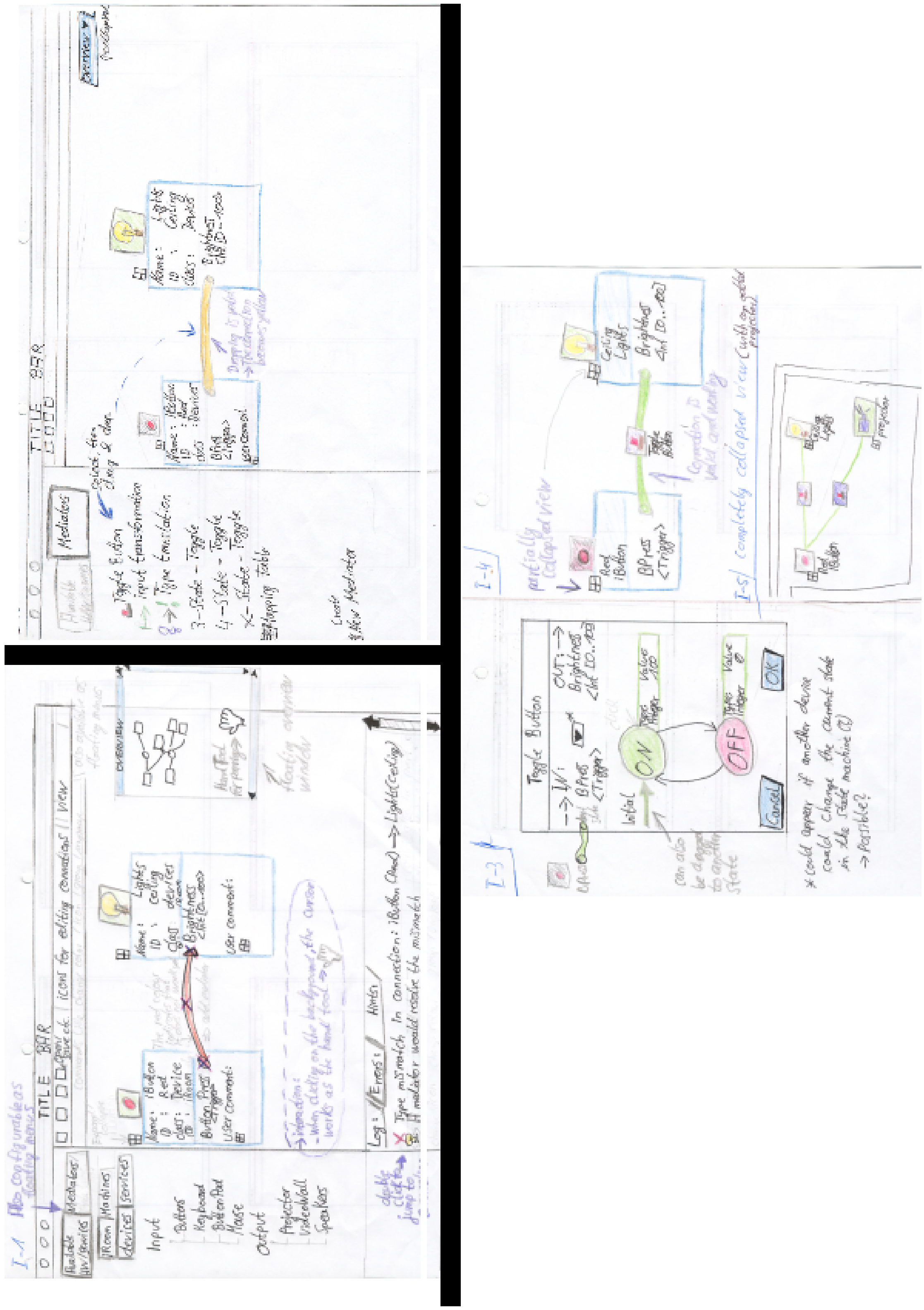


Figure A.1: The original storyboard created for the scenario described in section 4.2.2 in a larger view. Interaction steps were manually written into the storyboard.

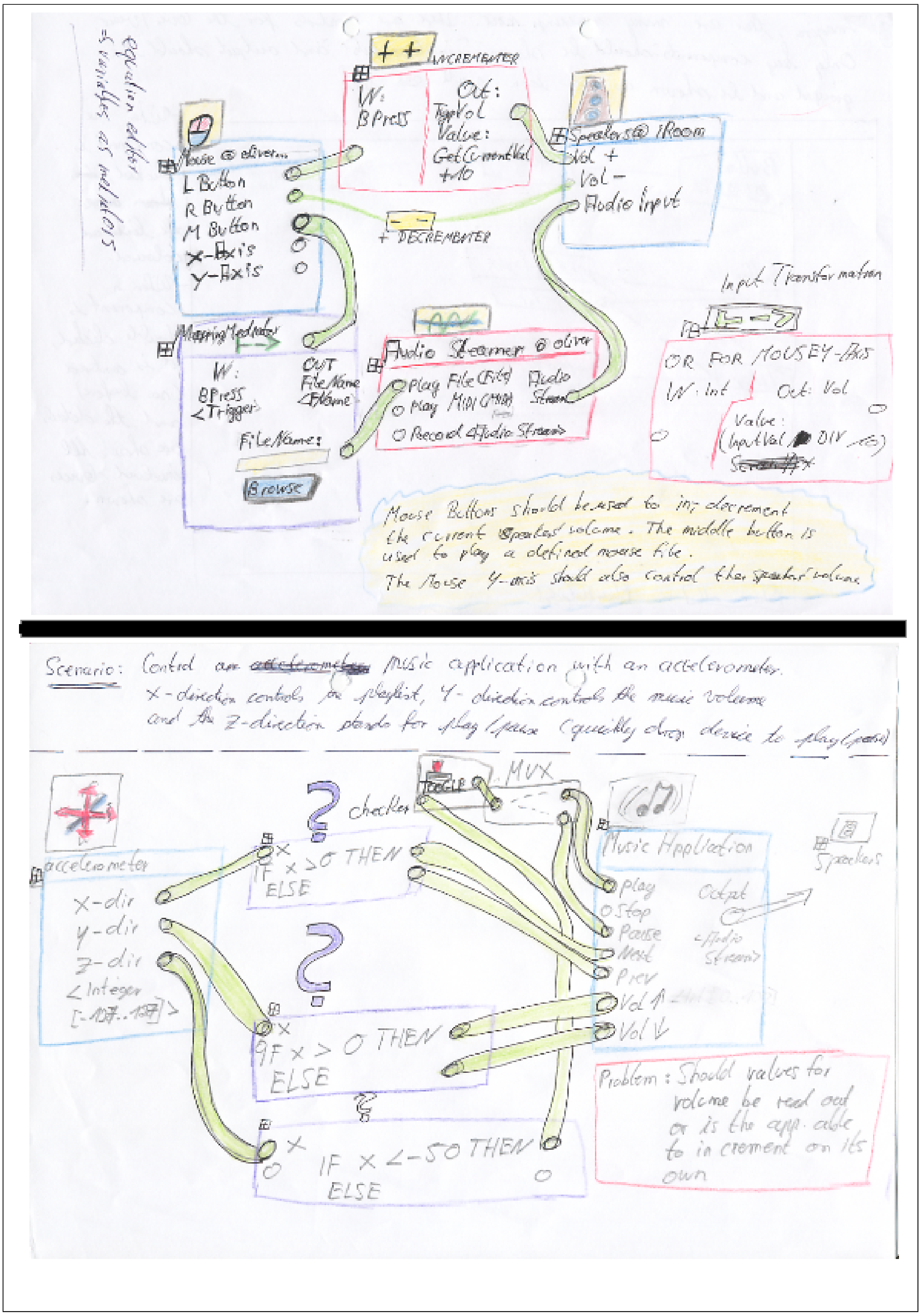


Figure A.2: This figure show two final result of two prototyping scenarios realized as a paper prototype for the Patch Panel GUI.

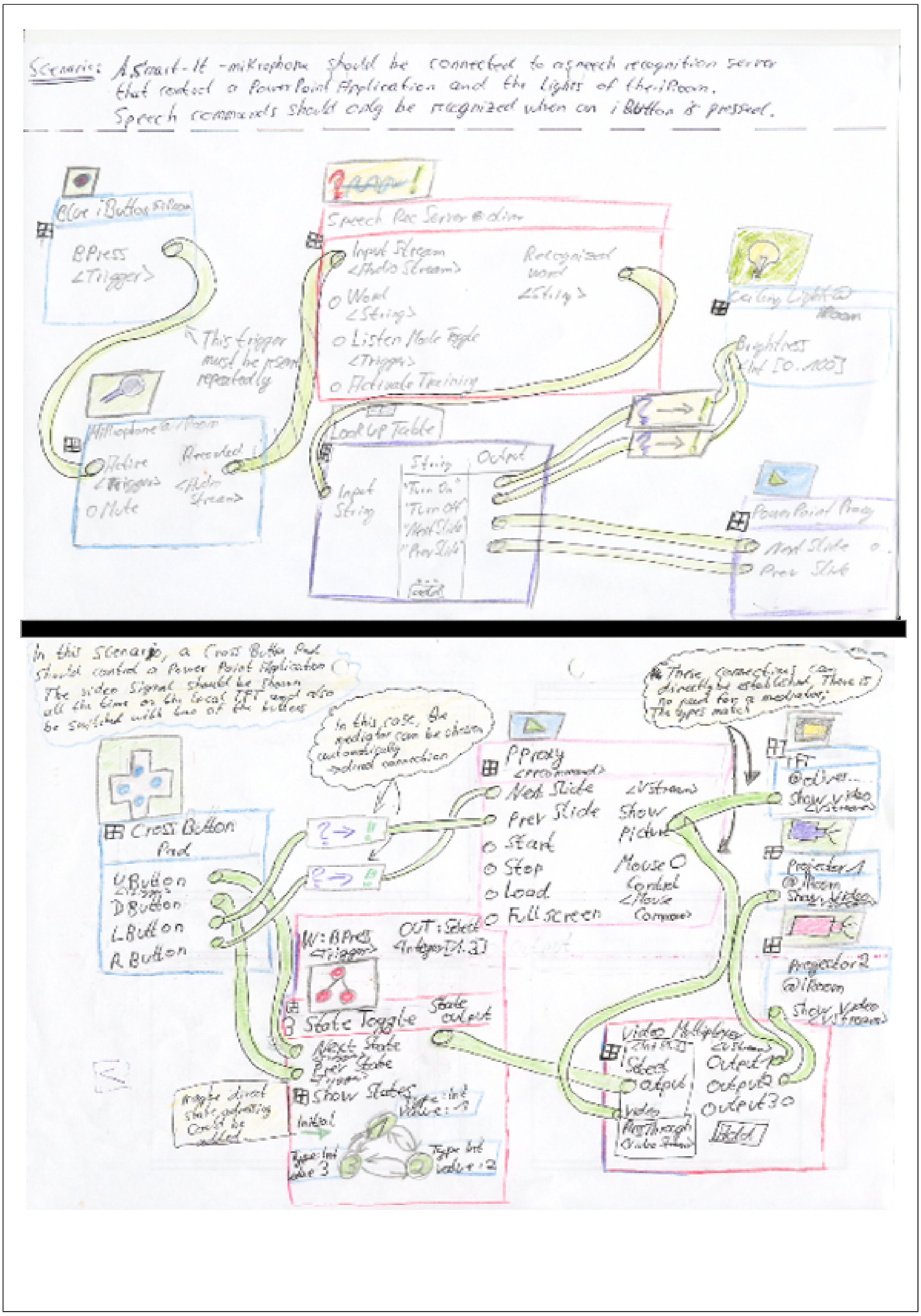


Figure A.3: Two more scenario drawings showing a working scenario created with the paper prototype version of the Patch Panel GUI.

Appendix B

Evaluation and scenario descriptions

User tests for the Patch Panel GUI and the Patch Panel Script language

Welcome to the user study of the iStuff project at the Media Computing Group and thank you for your participation. We are happy to give you an introduction into our latest development in the field of supporting the rapid prototyping process for ubiquitous environments: A new graphical interface for the *Patch Panel* and a command line wrapper responsible to facilitate the management of iStuff proxies, the *Proxy Manager*.

0. Important note

Important note on the test in that you take part for us: **It is not you to be judged or evaluated; it is our design of the software we hand to you!** So do not be afraid to criticize the development environment.
This lesson should include the benefit for you that you become familiar with the development suite and for us that we can justify a user evaluation and incorporate today's results of a user evaluation into our work.

The tasks you perform are recorded in terms of a screen capture and audio recording. Please do not feel offended by these techniques and just try to ignore them. We will not attempt to identify or judge your behavior with these recordings. They only represent a post-evaluation aid for us to see how you have derived a certain solution. You are not filmed explicitly. The iSight vision is turned off!

Figure B.1: Page 1 of description of the evaluation handed out to the participants - Introduction

1. Motivation

In order to give a motivation for the following scenarios we would like you to recreate certain design tasks. Please let your fantasy go a little beyond what you directly see and the context in which you perform your tasks.

Ithat you are part of a design team in a company that wants to produce a new type of mobile phones. The new design should include different sensors and explore new ways of interaction with the phone. Another research field lies in security systems as well as in controlling robotic assistance systems.

One week, your advisor enters your office and presents you different scenarios and asks you to evaluate them. The scenarios are presented in written form as you can see below. You take the papers he gives you and start thinking about different configurations and what types of input and output devices you will need.
With the help of the iStuff prototyping suite you want to build a first hardware prototype that can be shown to different test users. The results of these user studies should justify your current design or help to discover weaknesses or strengths that influence future designs. You know that your prototypes will look a little awkward but it is clear to you and also to your test users that the functionality matters, not the look of the device. If the interaction is well designed, a later product that integrates all the tested capabilities can be derived from that early design.

So, just go ahead and explore the possibilities of the *Patch Panel* and create whatever you think to be meaningful. And please remember: At that stage there is not right or wrong design, it has to be found out later. You just want to build something robustly working.

Figure B.2: Page 2 of description of the evaluation handed out to the participants - Motivation

Scenario 1: Control a multi screen presentation controlled via a mobile phone and Phidgets.

A presentation software running on two different machines should show the same set of slides. The screen of the left machine should display the last slide and the one on the right should show the current slide that is talked about. In order to control the slide transitions, the use of mouse and keyboard should be avoided. Instead, the presentations should be controlled via the presenter's mobile phone keys. An alternative to this design is the installation of pressure and touch sensors on the floor, the speaker has to step onto. A light sensor would also be possible.

Discover the event types of the other events through the Event Logger.
And also discover the mobile phone key values by examining the appropriate events on the Event Logger.

Scenario 2: Tilt to scroll (SmartIts sensor board)

Augment a mobile phone in the following way:
A SmartIts sensor boards is attached to the mobile phone.
When the phone is tilted upwards or downwards and a pressure threshold is crossed, the menu display should start scrolling in upward direction and downwards, respectively. Depending on the degree of tilting, the scrolling speed changes. In a first prototype, it should be differentiated between at least two scrolling speeds.

Figure B.3: Page 3 of description of the evaluation handed out to the participants - Scenarios 1 and 2

Scenario 3: Prototype a new iPod Shuffle using iTunes and Phidgets.

You are asked to build a new iPod Shuffle interface with a new generation of sensors. You decide to use Phidgets sensors to control the new version of the iPod Shuffle.
As a representation of the music player device, you use the iTunes application coming with your Mac OS distribution. An iStuff proxy that controls the application is available. Basic interaction such as Play/Pause, Next/Previous Track and the increase and decrease of volume are to be implemented in the scope of this scenario.

Scenario 4: Control motor with SmartIts and Phidgets).

An automatic balance control system for boats should be developed. Whenever the boat is tilting too much to the side, the motor should move a counter weight attached to it in order to balance it again.

Make use of the SmartIts Accelerometers and the Phidgets Servo Motor Controller for this scenario. As an extension, you could also make use of other extensions with Phidgets.

Figure B.4: Page 4 of description of the evaluation handed out to the participants - Scenarios 3 and 4

Appendix C

Post participation questionnaire

Post participation questionnaire

1. Would you say that the development capabilities with the Quartz Composer approach are better than with the script?

I strongly agree	I agree	Don't know	I disagree	I strongly disagree
( )	( )	( )	( )	( )

2. Could you imagine more scenarios where ubiquitous devices are connected and configured in the presented way?

I strongly agree	I agree	Don't know	I disagree	I strongly disagree
( )	( )	( )	( )	( )

3. How do you like the data flow metaphor of the Quartz Composer?

It was a strong help	It was useful	Don't know	It was no help	It absolutely made no sense to me
( )	( )	( )	( )	( )

4. Would you say that you used lots of the Quartz Composer original patches?

I used them a lot	I used them	equal to the new one	seldom	not at all
( )	( )	( )	( )	( )

5. The general concept of the iStuff project (Event Heap infrastructure, distributed exchange of information with events, etc.) was understandable.

I strongly agree	I agree	Don't know	I disagree	I strongly disagree
( )	( )	( )	( )	( )

6. How would you judge the future extensibility of the graphical approach?

Very extensible	extensible	don't know	hardly extensible	none at all
( )	( )	( )	( )	( )

7. I would prefer the **graphical approach** over the scripting approach

I strongly agree	I agree	Don't know	I disagree	I strongly disagree
( )	( )	( )	( )	( )

8. I would prefer the **scripting approach** over the graphical approach

I strongly agree	I agree	Don't know	I disagree	I strongly disagree
( )	( )	( )	( )	( )

9. Would you say the script approach is more flexible or powerful?

I strongly agree	I agree	Don't know	I disagree	I strongly disagree
( )	( )	( )	( )	( )

Figure C.1: Page 1 of the questionnaire handed out to the participants after the user test

10. What approach encourages you to go through many iterations?

Graphical Appraoch (Quartz Composer) ( ) Scripting Approach ( )

11. What would you like to be changed in future version?

12. What features would you add?

13. What patches were you missing during the prototyping process?

14. Additional comments: (Write whatever you think!)

15. How would you judge your former experience with Quartz Composer (1 = No at all, 5 = Quartz Composer expert)?

Figure C.2: Page 2 of the questionnaire handed out to the participants after the user test

Appendix D

Discussion of different implementations of a user test scenario

Figure D.1 shows four slightly different solutions for the music player scenario described in section 7.1.3.

Solutions for scenario 7.1.3

Solution a makes uses "JavaScript" patches only (cf. figure D.1a) whereas the group that created the second solution preferred conditionals to specify thresholds for trigger (cf. figure D.1b). The third solution even went further than the scenario description (cf. figure D.1c); The sensor inputs were also used for controlling graphical animations. In the last solution, the user group wanted to build in a "Math" patch, too but the time for them ran out in order to extend their solution.

Different solutions

From that example it can be seen that the groups tried out different ways to complete their tasks. It was also tried to explore the available patches inside the Patch Panel GUI in order to augment the solution.
One might argue that with more time for the setup, the implementation probably would have been refined.

Exploration and usage of existing patches

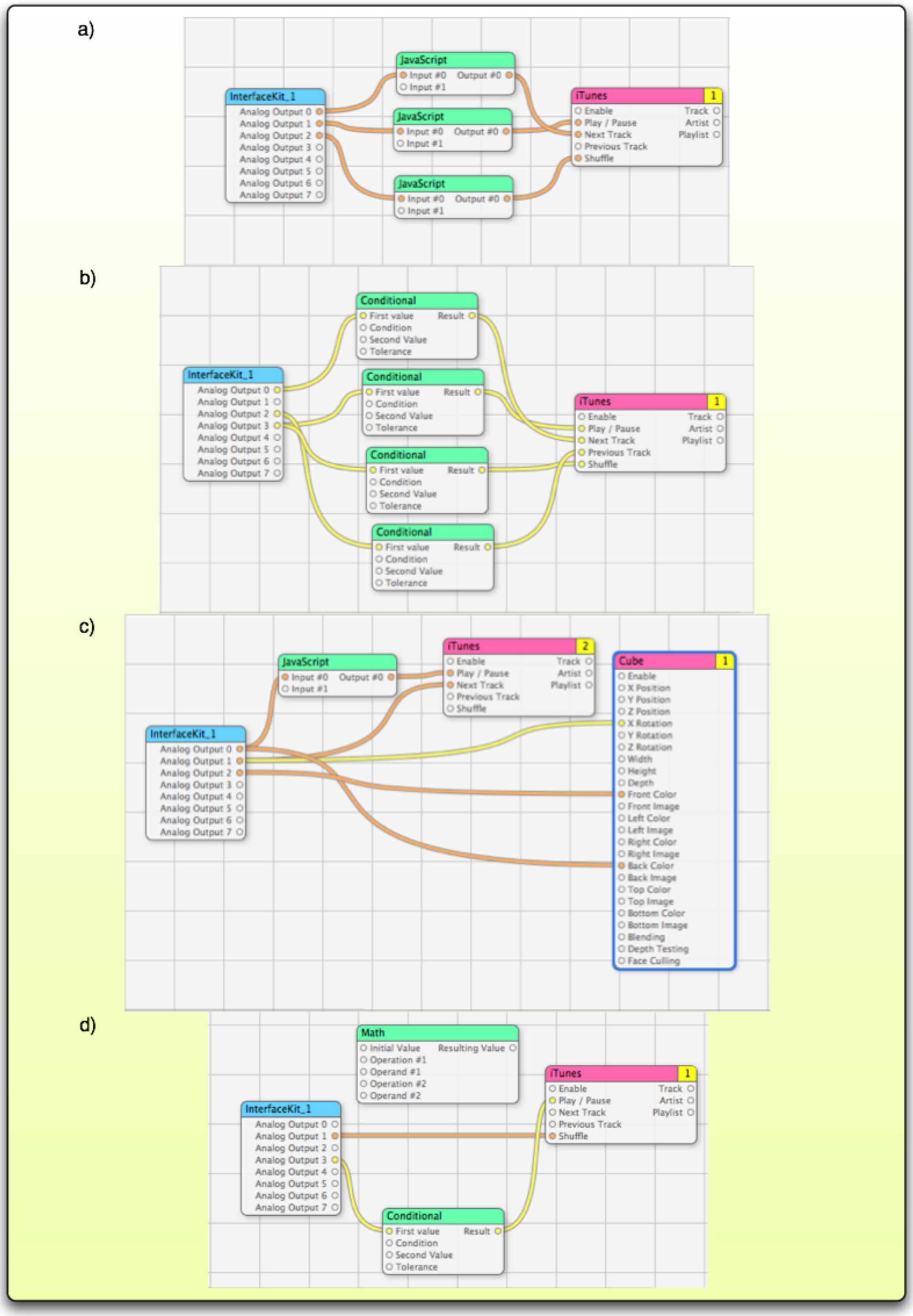


Figure D.1: The prototyping scenario for a new music player device was implemented in different ways

Bibliography

Rafael Ballagas. Patch panel: Distributed i/o management for ubicomp. *UBICOMP Doctoral Colloquium*, September 2004.

Rafael Ballagas, Meredith Ringel, Maureen Stone, and Jan Borchers. istuff: A physical user interface toolkit for ubiquitous computing environments. In *CHI 2003*. ACM, 2003.

Rafael Ballagas, Andy Szybalski, and Armando Fox. Patch panel: Enabling control-flow interoperability in ubicomp environments. In *PerCom 2004 Second IEEE International Conference on Pervasive Computing and Communications*, Orlando, Florida, USA, March 2004.

Rafael Ballagas, Michael Rohs, Jennifer Sheridan, and Jan-Borchers. Sweep and point & shoot: Phonecam-based interactions for large public displays. In *CHI '05: CHI '05 extended abstracts on Human factors in computing systems*, pages 1200–1203, New York, April 2005. ACM Press.

Rafael Ballagas, Faraz Ahmed Memon, René Reiners, and Jan Borchers. Rapidly prototyping mobile interaction in ubiquitous computing environments phone interactions in ubiquitous computing environments. In *Submitted to Ubicomp 2006*, Orange Country, California, USA, April 2006a. Submitted to UBICOMP 2006.

Rafael Ballagas, Faraz Ahmed Memon, René Reiners, and Jan Borchers. istuff mobile: Prototyping interactions for mobile phones in interactive spaces. In *Proceedings of PERMID '06*, 2006b. URL `http://www.medien.ifi.lmu.de/permid2006/`.

Eric Bergman and Rob Haitani. Designing the palmpilot: A conversation with rob haitani. In Eric Bergman, editor, *Information Applicances and Beyond*. Morgan Kaufmann, 2000.

Jan Borchers. *A Pattern Approach to Interaction Design*. John Wiley & Sons, 2001.

Jan Borchers, Rafael Ballagas, and Maureen Stone. istuff: Searching for the great unified input theory. *Ubicomp 2002 Workshop*, 2002.

A. Butter and D. Pogue. *Piloting Palm: The Inside Story of Palm, Handspring, and the Birth of the Billion-Dollar Handheld Industry*. Wiley & Sons, 2002.

W. Buxton. Lexical and pragmatic considerations of input structures. *Computer Graphics*, 17(1):31–37, 1983.

S. Card, J. Mackinlay, and G. Robertson. The design space of input devices. In *CHI 1990*, pages 117–124, 1990.

Tammara T. A. Combs and Benjamin B. Bederson. Does zooming improve image browsing? In *Proceedings of the fourth ACM conference on Digital Libraries*. ACM Press, January 1999.

Pierre Dragicevic and Jean-Daniel Fekete. Support for input adaptability in the icon toolkit. In *ICMI'04*, State College, Pennsylvania, USA, October 2004. ACM.

W. Keith Edwards et al. Using speakeasy for ad hoc peer-to-peer collaboration. In *Proceedings of CSCW '02*, November 2002.

Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison Wesley, 1995.

Beverly L. Harrison, Kenneth P. Fishkin, Anuj Gujar, Carlos Mochon, and Roy Want. Squeeze me, hold me, tilt me! an exploration of manipulative user interfaces. In *CHI "98: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 17–24, New York, NY, USA, 1998. ACM PRess / Addison-Wesely Publishing Co.

Björn Hartmann, Scott R. Klemmer, Michael Bernstein, and Nirav Mehta. d.tools: Visually prototyping uis through statecharts. In *UIST '05*, 2005a.

Björn Hartmann, Scott R. Klemmer, Michael Bernstein, and N. Metha. d.tools: Visually prototyping uis through statecharts. In *Extended Abstracts of UIST 2005*, pages 23–26, Seattle, WA, October 2005b.

Jan Humble, Andy Crabtree, Terry Hemmings, and Karl-Petter Åkesson. ”playing with the bits” user-configuration of ubiquitous domestic environments. In *Proceeding of the 5th International Conference on Ubiquitous Computing*. Springer Verlag, 2003.

Jan Humble et al. Configuring the ubiquitous home. In *Proceedings of the 6th International Conference on the Design of Cooperative Systems*. IO Press, May 2004.

Brad Johanson and Armando Fox. The event heap: A coordination infrastructure for interactive workspaces. In *Proceedings of the 4th IEEE Workshop on Mobile Computer Systems and Applications (WMCSA-2002)*, New York, June 2002. Callicoon.

Brad Johanson, Armando Fox, and Terry Winograd. The interactive workspaces project: Experiences with ubiquitous computing rooms [version no2, 4/11/02]. *IEEE Pervasicce Computing Magazine*, 1(3), April-June 2002.

Johnny Lee, Daniel Avrahami, Scott Hudson, Jodi Forlizzi, Paul Dietz, and Darren Leigh. The calder toolkit: Wired and wireless components for rapidly prototyping interactive devices. In *Proceedings of the ACM Symposium on Designing Interactive Systems (DIS'04)*, pages 141–146. ACM, August 2004.

John H. Maloney and Randall B. Smith. Directness and liveness in the morphic user interface construction environment. In *UIST 95*, pages 14–17, Pittsburgh PA, USA, November 1995. ACM.

Faraz Ahmed Memon. *iStuff Mobile: Rapidly prototyping novel interaction for mobile phones*. Department of Media Informatics RWTH Aachen, Aachen, Germany, 2006. URL `http://www-i10.informatik.rwth-aachen.de/faraz.html`.

Mark W. Newman et al. User interfaces when and where they are needed: An infrastructure for recombinant cmputing. In *Proceedings of UIST '02*, October 2002.

Jakob Nielsen. Iterative user-interface design. *Computer*, 26 (11):32–41, 1993.

Harmut Obendorf. The making of the palm pilot - refelections on a minimal information applicance. URL `asi-www.informatik.uni-hamburg.de/personen/obendorf/download/2005/pa_chi05.pdf`. Submitted to CHI 2005, 2005.

George G. Robertson, Mary P. Czerwinski, and John E. Churchill. Visualization of mapping between schemas. In *CHI 2005*. ACM, 2005.

Albrecht Schmidt, Kofi Asante, Antii Takaluoma, Urpo Tuomela, Kristof Van Laerhoven, and Walter Van de Velde. Advanced interaction in context. In *HUC "99: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*, pages 89–101, London, UK, 1999. Springer Verlag.

Timothy Sohn and Anind Dey. icap: Rapid prototyping of context-aware applications. In *CHI 2004 ACM Conference on Human Factors in Computing Systems*. ACM, ACM Press, October 2003.

K. N. Truong and G.D. Abowd. Inca: A software infrastructure to facilitate the construction and evolution of ubiquitous capture & access applications. In *Proceedings of Second International Conference on Pervasive Computing (Pervasive 2004)*, pages 140–157. Springer Verlag, 2004.

K.N. Truong, E.M. Huang, and G.D. Abowd. Camp: A magnetic poetry interface for end-user programming of capture applications for the home. In *Proceedings of the 6th International Conference on Ubiquitous Computing (Ubicomp 2004)*, Nottingham, UK, 2004.

Douglas K. VanDuyne, James A. Landay, and Jason I. Hong. *The Design of Sites*. Addison-Wesley, 2002.

Mark Weiser. The computer for the 21st century. *Scientific American*, 265(3):94–104, 1991.

Eugen Hon Wai Yu. *Evaluation of Focus Methods in a Ubiquitous Computing Environment*. Department of Media Informatics RWTH Aachen, Aachen, Germany, 2006.

Index

Typeset May 26, 2006