MASTER THESIS

TITLE: Ubicomp: Using iStuff

MASTER DEGREE: Master in Science in Telecommunication Engineering & Management

AUTHOR: Josep Benavent Marco

DIRECTOR: Roc Meseguer Pallarès

DATE: 8th June 2010

Title: Ubicomp: Using iStuff

Author: Josep Benavent Marco

Director: Roc Meseguer Pallarès

Date: 8th June 2010

Overview

Ubiquitous computing, ubicomp, represents a scenario where computer devices are omnipresent usually with a look like not traditional computers. For this, is also known with the term “disappearing computer”.

There exists also a close relation between ubicomp and human to computer interaction. With the appearing of new computer devices spread along the environment, new interfaces to resolve human to computer interaction.

This master thesis presents a ubiquitous computing scenario that uses iStuff toolkit as communication path. Nintendo’s WiiRemote, known as wiiMote, is used as new human to computer device to allow user’s interaction. WiiMote is communicated with a computer using WiiGee java libraries that incorporates an iStuff proxy that lets communicating with the scenario. The scenario also includes a collaborative application, called Collaborative Tetris, to interact with.

The first chapter presents technical concepts and tools: Starting with definitions of terms ubiquitous computing and human to computer interaction concept and then the main tools used to develop this master thesis: iStuff toolkit, WiiMote and WiiGee java libraries

The second chapter describes the start up of mentioned tools and the description of test scenarios.

Third chapter summarizes the test results of the scenarios: iStuff start up scenario, communication between iStuff and WiiMote using wiiGee and a possible ubiquitous computing environment with a collaborative application: collaborative Tetris.

Finally the conclusions of tests, possible effects over environment in a green study and personal conclusions are present.

INDEX

INTRODUCTION

Nowadays, technology bases its relation with humans in such a way that computers are still the focus of attention. In other words, you have to learn how to communicate with the computer using its own peripheral equipment as keyboard or mouse to success in your work. In a ubiquitous environment is the technology who, spread along the environment, tries to adapt to user's demands. In those environments, small computer devices try to interact with other small computer devices to facilitate the users' use of technology. This new paradigm in which technology tries to disappear from user's interaction is in what ubiquitous computing works.

As I introduces before, derives the idea that exists a thin relation between ubiquitous environments and new human to computer interaction (HCI): meanwhile technology "disappears" new human centered devices have to appear to solve the missing gap between the old peripheral equipment, centered on computer, and the new generation of equipment centered on user's attention.

Writing this master thesis I try to introduce myself into the ubiquitous computing environments and by extension, obtain a first approach into new human to computer interaction devices. The reason: ubiquitous environment are still in a premature state and I think the future of technology will go through this kind of devices and in a short term, we will start to enjoy this in our house's environments.

The main objectives proposed are:

- Determine capabilities of wiiGee library for detecting different gestures and capabilities of iStuff architecture to create a ubiquitous test environment.
- Creation and integration of an iStuff proxy in wiiGeeGUI.
- Create a ubiquitous test environment in which all the parts can interact together.

This master thesis is structured in four chapters with the following contents:

Chapter one describes the technical concepts and tools that are used to develop the master thesis. First a definition of ubiquitous computing is present. Then a short description of some relevant projects into the ubiquitous computing environments are done, focusing more attention in iStuff project, which technology is used later to develop a ubiquitous computing environment. Also a short introduction to new human to computer interaction paradigm is done with the description of the term, the description of the Nintendo's Wii remote, as a new peripheral equipment as example of a device not centered in computer, and the description of the software that permits the interaction with the computer of that device, WiiGee java libraries. You also can find a description of collaborative applications, because it is an example of application where new HCI can be emphasized.

In the second chapter you can find the start up description of iStuff libraries, WiiGee libraries and how to connect successfully a WiiRemote device to a computer. Also a description of the proposed tests scenarios can be found to prove that all is working.

The chapter three summarizes the results of the tests proposed on the second chapter. The first one shows the success working of istuff library, the next one the interaction of wiiGee libraries with the iStuff event bus trying to control an operating system remotely. Finally, a complete scenario in which both libraries are used to interact with a collaborative application: in this case, a collaborative Tetris is created to show the final result of the interactivity among all different parts.

Chapter four presents the conclusions of this master thesis. First specific conclusions about tests results are picked up and then technology specific evaluation conclusions. In this chapter you can also find my personal conclusions about my experience conducting this master thesis and finally, a green study about the environment impact of this thesis.

Finally some annexes are included with source code that is created or modified from examples. Annex I includes the source code used in the first test, annex 2 the source code involved in the second test and Annex 3 the source code that is referred in the third test.

CHAPTER 1. TECHNICAL CONCEPTS & TOOLS

This chapter is firstly to familiarize the reader with the “ubiquitous computer” term and related vocabulary used in this thesis and exposes some examples to facilitate the reader’s understanding. Then the main tools that are used to develop the project are introduced: iStuff toolkit as a software platform that acts as an event bus and the Nintendo’s WiiRemote as a human to computer interface to interact with that bus using the WiiGee java libraries that interprets WiiRemote gestures.

1.1. Ubiquitous computing:

In this section a first definition of the term is presented. Afterwards a short introduction to ad-hoc networks and finally some relevant projects are referred to facilitate the understanding of the explained terms.

1.1.1. Definitions:

The two following paragraphs illustrate by the use of examples the term “ubiquitous computing”, also known as ubicomp:

Imagine that you make a photo with a new camera. Once is done, this is send automatically to your personal computer and also tagged and edited with the time and date that the photo is done. Also, information about your location is transmitted by the GPS device that is automatically detected near your position and is also included into the photo tags to obtain a better classification of it. You only pressed the photo button; all other functions are done automatically.

Once you arrive to your office, you start your session automatically with your fingerprint in any of the personal computers you can find there. You received by e-mail a new catalogue from your suppliers and decide to print it. You say: “Print this document in the nearest printer”. Automatically, the document is sent and printed in the printer that is at your back without much more effort.

The terms ubiquitous computing is usually related with human-computer interaction and ad-hoc network terms. First of all, we are going to explain the meaning of these terms[1].

The Oxford English Dictionary [1] defines the “ubiquitous” as:

Present or appearing everywhere, omnipresent.

[1] Exists a dilemma about what is ubiquitous and what isn’t. I will try to expose a general idea about the term and not take into the view all specific definitions.

Ubiquitous computing refers to a scenario where computers are omnipresent, and specially related to devices that do not look like traditional computers. The idea that is behind this term is also well known as the "disappearing computing".

Frank Stajano, explains the meaning of this idea in the introduction of his book about security issues in ubiquitous computing environments [2]:

"When we use a computer, the focus is still squarely on the tool rather than on the task (...) Ubiquitous computing is not so much about having traditional computers everywhere, but rather about having computing capabilities everywhere, embedded in the environment in such a way that they can be used without noticing them. Computing will become ubiquitous when it supports the user's activity unobtrusively, instead of being the focus of attention."

Stajano also includes another explanation and makes a parallelism with the issues related to electrical motors when they were invented. Initially only one electrical motor could be found in the factories. Only this motor powered machines and mechanisms for a wide variety of tasks. Nowadays, the electric motor leads the daily life of every human and humans are unaware of the details of how this work and we unconsciously use it. This is a simple example of a "disappeared" technology that has succeeded in the history.

One of the objectives of ubiquitous computing is to adapt the human-computer interaction in such way that computers become, as I mentioned, a disappeared technology. The human-computer interaction (HCI) [9] but, is a multidisciplinary field in which we can include subjects as, engineering, design, anthropology, sociology, philosophy, linguistics, artificial intelligence, computer science, neuroscience or cognitive psychology that studies the relations between people (the users) and the computers. The goal of HCI is to improve the interactions between users and computers through developing interfaces (hardware or software), methods to develop that interfaces and techniques for evaluating them with the objective of making computers more usable and more receptive to the user's needs.

Defined the term ubiquitous computing and the HCI term, there's still an important term usually close to the ubiquitous computing term. This is the "ad-hoc networks" that describe how this particular "computers" can communicate each-other. We are going to define briefly this concept.

Ad-hoc, by de OED [1], means:

Devoted, appointed, etc…, to or for some particular proposes.

The idea behind the ad-hoc networking is that instead of being a permanent infrastructural fixture behind the communication devices, each device is capable of discover and recognize each other's presence and adapt itself to new configurations that can appear in the scenario.

1.1.2. Relevant projects

There exist a high number of projects related with the ubiquitous computing term and it is not my objective to take into account all of them in this section. All the following described projects on the following lines, from my point of view, provide a wide vision on ubiquitous computing and served myself to prepare the following chapters.

1.1.2.1. Active badge system

This project by ORL/AT&T Labs from Cambridge appeared in early 90's which proposed a solution for a location system indoor. The main idea is that a person could be localized into a building to receive, for example, incoming phone calls to his actual location [3].

The badge is composed by an infrared transceiver that sends periodically a unique code, which identifies the person who owns the badge, to a network of sensors placed around the building.

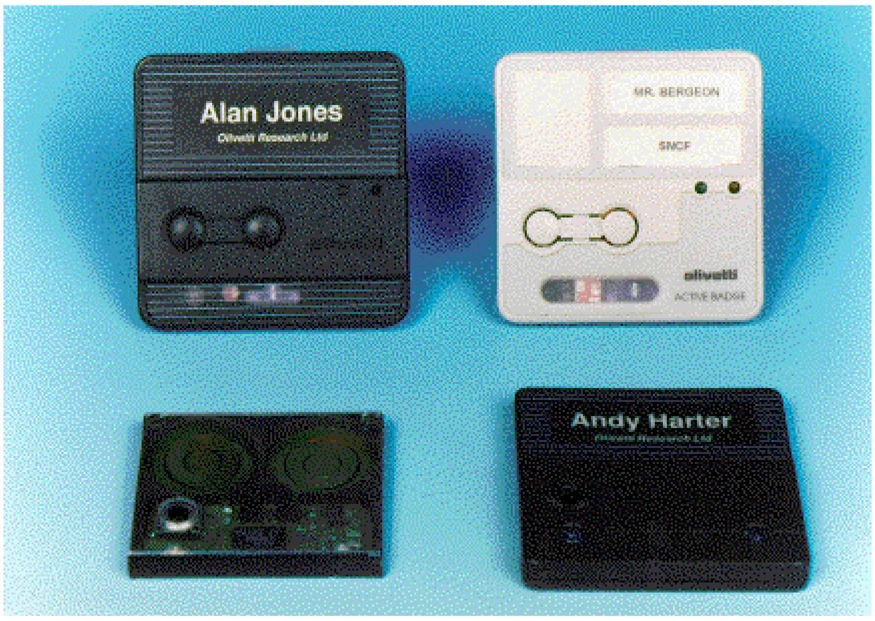


Fig. 1 Active badge System

The system also includes a user interface to obtain with a great precision the localization of a person inside the building.

Of course, the project soon obtained other applications like locate shared equipment, auto select the preferred beverage from a coffee machine for the person who is near it and also to configure a VNC service to obtain automatically your desktop in any computer you were in front.

1.1.2.2. Active bat

Another project by ORL/AT&T Labs from Cambridge that was started in 1995 and consists in a lightweight wearable devices featuring ultrasonic transducers, a 433Mhz radio transceiver and a long-life battery. Each device includes a unique identifier that identifies his owner or the object which is tagged to [4].

It was used throughout all the Cambridge Research Group building to precisely locate people and objects in a 3D environment.

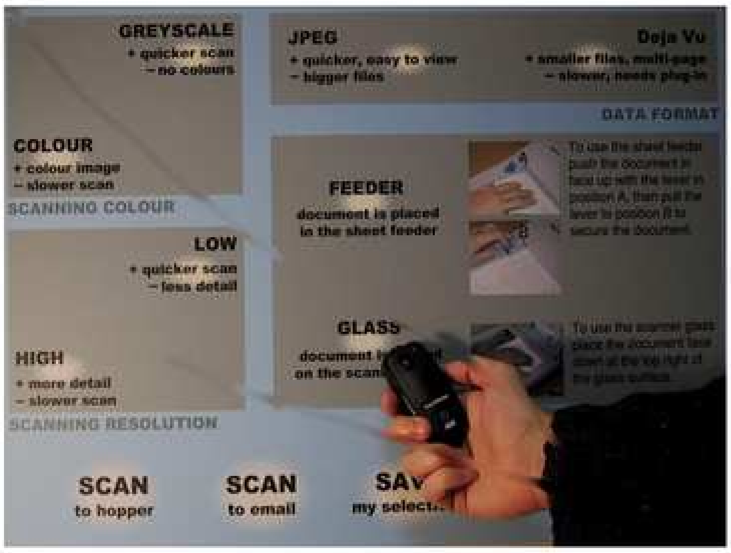


Fig. 2 Active bat example of use

Due to his high resolution (about 5 centimeters) it was used to define virtual buttons in a 3D environment. For example, as we can see in the previous photo (Fig.2), an interface was developed to interact with a public scanner can be implemented with an active bat system.

The system works as if the device was a 3D pointer. Depending on the position in front of the poster you place it, a related action is taken. When you place the pointer in front of an image, the position of pointer is processed and the action predefined on the poster is taken.

1.1.2.3. Oxygen

Oxygen[5] promises to be a world in which computation is available anywhere and at negligible cost. Is also focused in the human to computer interaction; nowadays, this interaction is computer based and the idea is evolve to a user based design[2] [6] or human-centered computing [7].

[2] User based design is a design philosophy that tries to optimize the users interface avoiding users to change how they work, rather than force the users to adapt themselves to computer interface, as nowadays occur.

This wide project includes a high number of components, as software and hardware, to obtain a complete solution for ubicomp environments. It includes speech recognition systems, image processing systems, user localization systems and sensors to support all these subsystems.

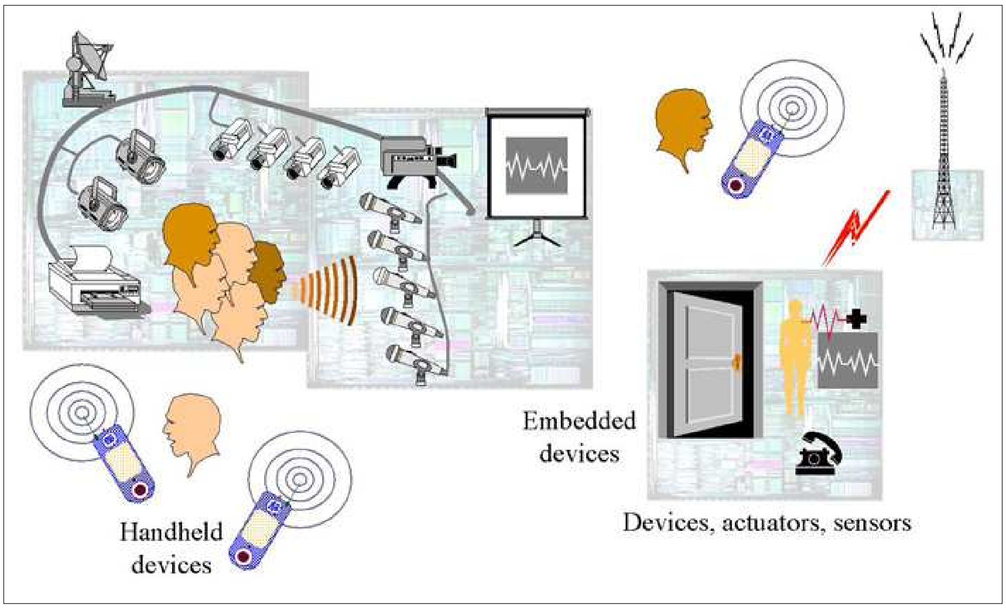


Fig. 3 Oxygen project overview

The components can be classified in the following three kinds of entities:

H21s: handheld devices acting as PDA'S, personal communicators (devices that includes cell phone functions, radio, pager, email …) and universal remote control units.

E21s: Computer server embedded in the environment (walls of houses, cars…) capable of affecting and controlling that environment. For example controlling lights, air conditioning systems…

N21s: Dynamic networks allowing the previous classes of devices to discover each other's services and make useful for them.

1.2. Collaborative applications

Also known as collaborative software or groupware applications, we can define the term from a general point of view [19]:

"A computer application that allows its users to collaborate with each other"

There exist also a large number of definitions and usually compared with the traditional software term, in which users are separated and don't feel part of a group. Other take into account not only software components, hardware can be part of collaborative applications.

As a complete definition we can achieve the following:

Collaborative application is a software and/or hardware application that:

- Interacts with multiple users: It receives input from multiple users and creates output for multiple users.
- Links these users: It allows some input of some user to influence some output created for some other user.

Note the output that is generated by users input has not to be immediately in time and individual; a group of inputs can generate an individual output.

1.3. iStuff Toolkit

Nowadays, the mouse and the keyboard are the predominant input devices for desktop computers. This desktop computer environment is targeted for one user, one set of hardware and a single point of focus. In a post desktop, ubiquitous computing (ubicomp) environments complexity increases: multiple displays, multiple input devices, multiple applications and multiple concurrent users.

iStuff toolkit, developed by Rafael Ballagas from Computer Science department in Stanford University, is developed to support user interface prototyping in ubiquitous computing environment [9], [10].

This project, from Stanford HCI Group developer around 2002, includes a toolkit of physical devices and a flexible software infrastructure. The objective of this project is to simplify the exploration of novel interaction techniques in the post-desktop era of multiple users, devices, systems and application collaborating in an interactive environment.

The environment is a TCP java based middleware that allows multiple machines and applications to exchange information through the Event Heap, a central server process that receives events from client applications and redistributes them to the appropriate recipients.

We can distinguish between the two main components of iStuff project: the event heap and the patch panel.

The Event Heap is a distributed system infrastructure that allows multiple users; machines and applications where all simultaneously interact as consumers and generators of system events.

The Patch Panel is a service that runs in the background and observes the Event Heap in order to translate events. Every event that is posted to the Event Heap is a translation candidate. If a mapping is specified for an event, the Patch Panel will post the corresponding output events to the Event Heap. This functionality is especially useful for prototyping new input or output devices and interfacing these devices to different applications.

1.3.1. Architecture

iStuff toolkit environment is developed with the following requirements:

- Flexible, lightweight devices.
- Platform independence and cross-platform capabilities.
- Wireless protocol independence.
- Ease of integration with existing applications.
- Support for multiple simultaneous users.

The iStuff components provide the physical toolkit of wireless input and output devices, asynchronous communications based on Events and the PatchPanel intermediary to dynamically re-map events to applications. The architecture can be summarized in the following Fig.4:

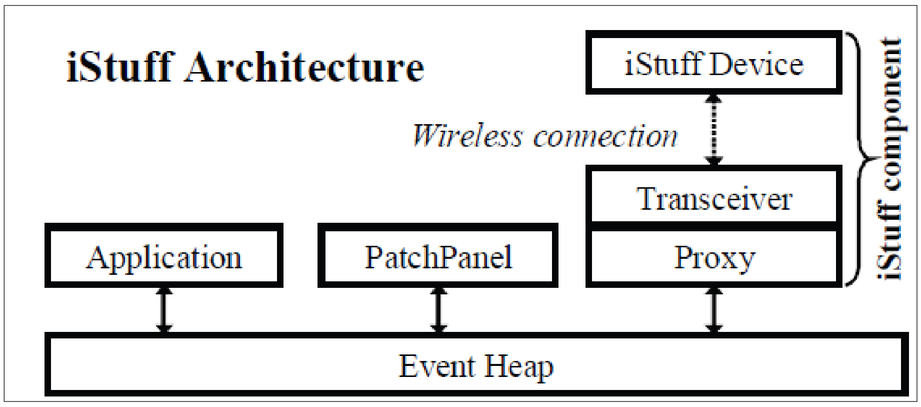


Fig. 4 iStuff architecture overview

iStuff Architecture is based on an Event Heap that acts as an event bus that receives the events created by an iStuffDevice that have been send by the iStuffDevice through an iStuff Proxy. Then the Patch Panel dynamically re-maps those events to another predefined user event and is send again to the Event Heap. Finally an application or other iStuffDevice Proxy also connected to the bus to act appropriately to that re-mapped event.

1.3.2. Components

Now, we will describe the iStuff architecture components in detail. The components of iStuff architecture are the following:

- **Event Heap:**

  The communication with the different iStuff components is done by event messages. Producers (that can be proxies or applications directly) post events to the Event Heap and consumers register to receive events, specifically the event type and optionally, other criteria based on matching the content of specific fields. This communication mechanism allows an easy restarting of any component of the architecture (including the Event Heap itself).

  The main function of Event Heap is permitting the exchange of events between event consumer and producers, acting as a bus.

- **Patch Panel:**

  The Patch Panel consists of an intermediary application that implements event mapping, and one or more GUIs that provide a user-accessible way to configure events.

  It acts as an Event Heap client that non-destructively translates events from one type to another. It listens for all events, translating those that match its event-mapping configuration. To create a mapping through the intermediary, the users must generate an event of type IntermediaryConfigEvent with the appropriate fields that represent the event to translate and its mapping. When the Event Heap receives a new IntermediaryConfigEvent, it updates its internal translation look-up structure.

  For example, the simplest event mapping matches only the event type, generating another type of event from the received. Another common mapping, matches the event type and a unique ID field, for example, to discriminate between two same models of input devices.

  Patch Panel component also includes a GUI that can be placed in another machine connected to the Event Heap or in the same machine. This allows a rapid prototyping of new devices. It presents the user with a graphical tool for creating event mappings. After the user specifies a particular event translation, the Patch Panel GUI posts an IntermediaryConfigEvent to update the Event Heap.

  Another highlight point is the possibility of create composite events establishing intermediate stages using event translations in the Patch Panel. For instance, an application could be notified if two iButton where pressed. Patch Panel can create an intermediate stage between the inputs devices are activated in a short period of time. When both are finally pressed, the system can create the predefined event and reinitialize the stages to the initial state.

Finally, and to facilitate the developer test the environment is provided with a powerful characteristic; you can compose scripts and load to the Event Heap using that PatchPannel GUI, to create more complicated “logic” with the events mapping and configuration.

```
program Demo;

event Start type Demo {
   string message = "Hola";

}

event Saluda type Demo {
     string message = "Que tal?";
}

initial state Initial {
   on Start {
   send Saluda;
   }
}
```

On previous lines, as example, you can observe a basic script that receives a packet type Demo with a string field called “message” containing the message “Hola”. If the scrip is loaded in the Event Heap, each time that this defined event is send, a packet type Demo is send with string filed called message with the value “Que tal?”.

- **Proxy:**

  The iStuff proxy component allows communication between physical devices through the Event Heap. This proxy is the component that translates the events from the physical world (for example the pushing of a button) to the iStuff events environment understandable message: an event. It also does the inverse action; collect the appropriate events that “surf” the event Heap and translates it to the attached device with an understandable iStuffDevice action or message.

  The proxy component also provides of independence between the proxy-iStuff device link technology and the Event Heap. For example, iStuff device can use FM or X10 technology to connect with the proxy element and this proxy use Ethernet wired link with the system Patch Panel.
- **iStuff Device:**

  Stanford University research group who developed iStuff toolkit created a high number of devices with his respective proxy component. These devices can act as inputs or actuators and thanks to the proxy element, are independent of link technology. As basic example, some are briefly described on following line, and in Fig.5 can be seen:

- iButton: The most basic one. It is implemented using a homemade circuitry and a garage-door-opener style radio frequency transmitter. Another example is implemented using commercial X10[3] keychain remotes. This is an input device.
- iLight, iBuzzer and iViber: Binary output components or actuators implemented using homemade circuitry and RF transmitters. They can provide a visual (iLight), audio (iBuzzer) or a haptic (iVibe) output.

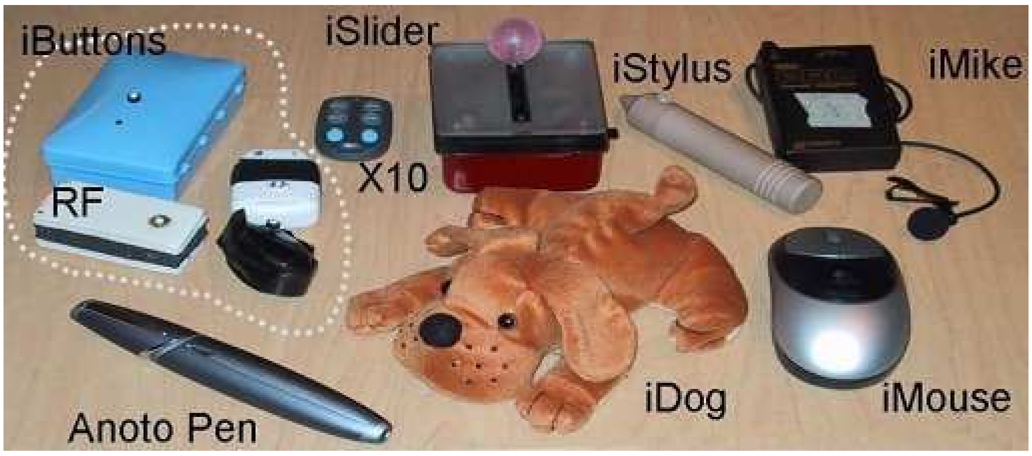


Fig. 5 iStuff devices

1.3.3. Applications

iStuff was originally designed with iRoom (space used for meetings and brainstorming/design session in Stanford University) but it could be useful in Smart Home environments. Task-oriented user interface (interfaces reflecting the user's task as opposed to the technical feature for an appliance) is the highlight point to be useful in Smart Home applications:

- **Dynamic, task-based remote controls**: By linking the appropriate iStuff components and using the Patch Panel the user can construct a task-oriented controller. For example, a PDA can be used as a Universal Remote Controller.
- **Monitoring house state**: A display near a user's home exit door can show the state of linked iSensors to home devices. For instance, the state of burglar alarm or if the lights in the bedroom are still on.
- **Setting house state**: iButton or a similar device can be configured to set the house state when user leaves it. For example, when the button is

3 [12] Open industry standard for communicating electric devices. Uses power line wiring for signaling and control. Also a wireless radio based transport protocol is defined.

pressed the system switch off all the lights or activate the home security system.

1.4. Wii Remote

The Wii Remote or “WiiMote” is the primary controller for Nintendo’s Wii console. The main feature of the Wii Remote is its motion sensing capability, through the use of accelerometer and optical sensor technology. It’s able to use a Sensor bar to use a pointing functionality. It also includes a rumble that when activated it will cause the controller to vibrate.

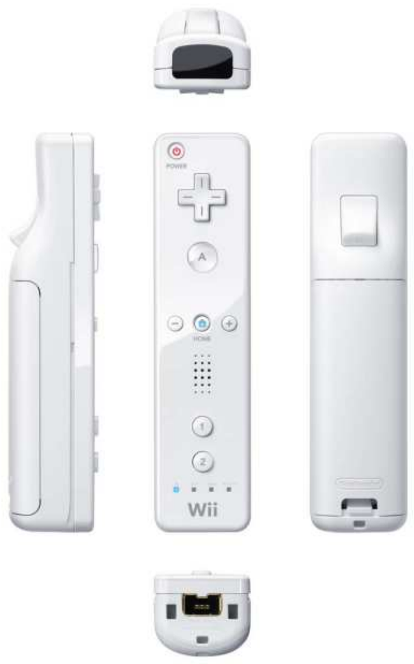


Fig. 6 Wii Remote Controller Overview

The Wii Remote is a one-handed remote control-based design. The body of the controller measures 148 mm long, 36.2 mm wide and 30.8 mm thick. The communication with the console device is via short-range Bluetooth radio protocol, with which it is possible to communicate up to four controllers as far as 10 meters. To use the pointing functionality a Sensor Bar is required and is able to work up to five meters. His symmetrical design allows it to be user in either hand and also can be turned horizontally to be used as the classic Nintendo controllers.

It includes nine buttons: POWER, A, B, -, HOME, +, 1, 2 and the digital path controller (UP, DOWN, LEFT and RIGHT). We can also find a speaker (low-quality 21mm piezo-electric speaker used for sound effect during game play and streamed directly from host) in the center and four leds in the below that indicate various functions.

An expansion port can also be found at the button of controller to be used with extension controller such as “Nunchuk” or a Classical controller.

The power source are two AA class batteries with you can obtain a life cycle of 60 hours using only the accelerometer functions and only 25 hours if you also use pointing functionality.
The Bluetooth device is model *BCM2042* by Broadcom’s manufacturer [20] and the accelerometer is an *ADLX330* chipset from Analog Devices manufacturer [21].

The rumble, which is implemented as a small motor attached to an off-center weigh, It can be different implemented using different motors, *SEM8728* device is an example. As the Wii Remote drives at 3.3 VDC and draws 35 mA, other rumble device complaining these requirements can be used.

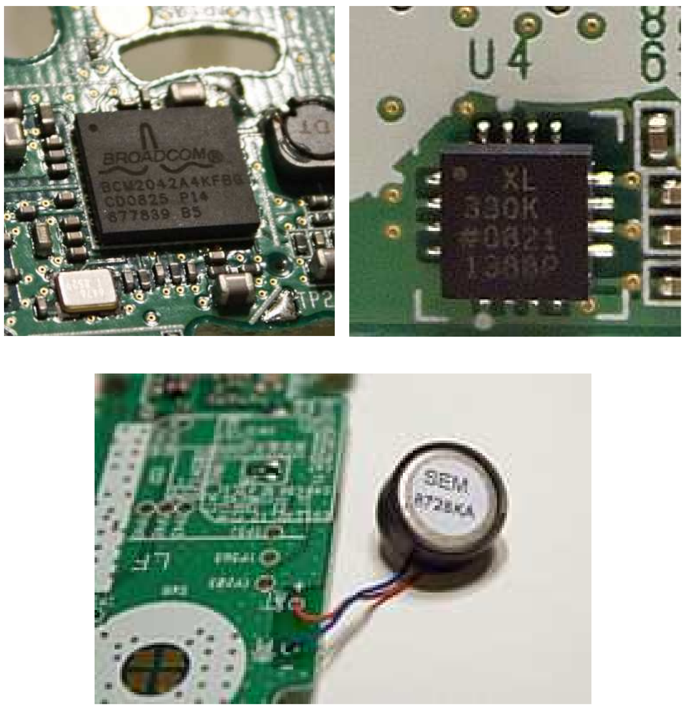


Fig. 7 Broadcom BCM2042, Analog Devices ADLX330 and SEM 8782 detail.

1.5. WiiGee Library

Wiigee [14] is an open-source gesture recognition library for accelerometer-based gestures specifically developed for the Nintendo Wii remote controller. It is implemented in Java and, thus, is platform-independent. Using a third-party Bluetooth-library wiigee allows you to define and recognize your own, freely trained gestures.

WiiGee library is based on detecting *gestures* in two or three dimensions. The gesture term is typically defined as a characteristic pattern of incoming signal data. Due to Wiimote is always sending information of the accelerometer behavior; wiiGee defines the start and the end of a gesture by pressing one of the wiimote's buttons.

The pipeline that wiigee follow to recognize gestures are explained following and the fig. 8 show the system components:

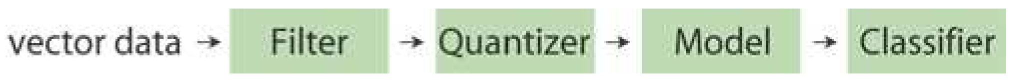


Fig. 8 WiiGee system components

First, the vector data is send by wiGee Bluetooth interface to the wiiGee core. Then these data is processed by the following components [22]:

- <u>Filter:</u> Removes vectors which are below a given threshold of approx. 1.2g (where g is the acceleration gravity) and removes vectors which are suspected to be duplicated

- <u>Quantizer:</u> Generates a discrete code identifying each gesture. K-mean algorithm is used, k being the number of clusters in our code-alphabet, using k=14 a sufficient size for the code alphabet is taken.

- <u>Model:</u> A discrete, left-to-right hidden Markov model. This component maps a series of codes to a model probability.

- <u>Classifier</u>: Identifies the probability model with the most probable gesture during a recognition task.

Due to wiiGee don't specify default tasks and let you to define your own *gestures*, two working modes are defined. The following Fig.9 shows the workflow:

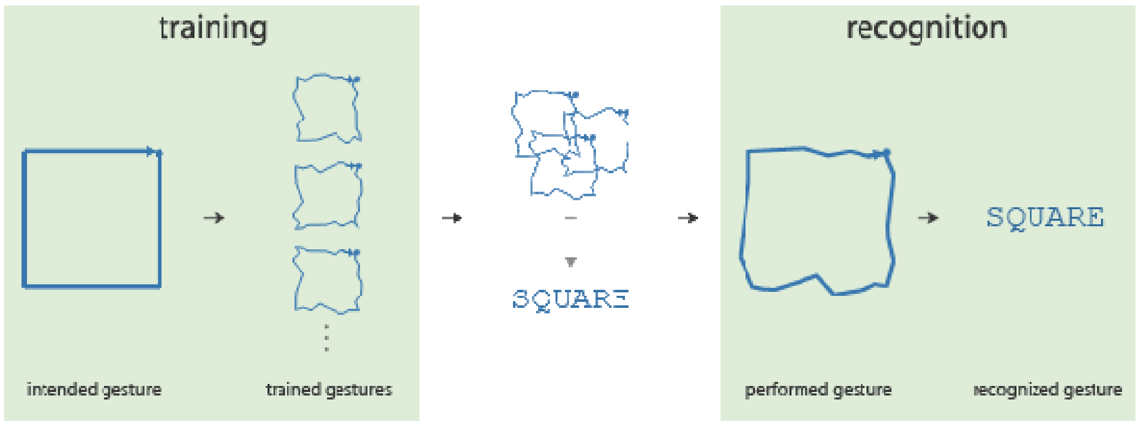


Fig. 9 WiiGee workflow

- <u>Training</u>: During this mode you record the *gestures* your application later recognizes as input commands. A gesture must be performed several times (wiiGee's developers recommend between 5 or 10 training sessions) to allow wiiGee learn the *gesture* ant internally code it. To facilitate user's training mode a training button is defined to enter and exit training mode.

- <u>Recognition</u>: Once gestures have been recorded, wiiGee internally sets up its parameters for the recognition process. Also a Recognition Button is defined; when it is held down the system "records" your movement and when released wiiGee tries to identify the gesture and fires a *GestureEvent* containing information about the detected *gesture* with its calculated probability.

CHAPTER 2. Technical implementation

As starting point for this project, we have at one's disposal the following tools that are described in detail in the previous chapter:

- iStuff java library.
- Wii remote or WiiMote, used together with
- WiiGee java libraries

In this chapter you can find the start up description of the mentioned parts and the description of the scenarios that are arranged to test all the parts individually and in group.

2.1. iStuff java library

iStuff toolkit most recent components can be found on the SVN: http://svn.berlios.de/svnroot/repos/istuff/trunk

To download it from this repository a SVN client can be used. For example, TortoiseSVN is a free software that can be found on http://tortoisesvn.tigris.org/.

After downloading the most recent version from the repository, something similar can be found on your local hard disk:

```
../istuffToolkit/
        . / Event Heap
        . / Event Logger
        . / Hardware Proxies
        . / installer
        . / Lib
        . / Patch Panel
        . / Patch Panel Manager
        . / ProxyLauncher
        . / ProxyManager
        . / QuartzComposer
        . / Readme.txt
        . / Software Proxies
```

To start a first demo using the toolkits the following step has to be taken:

1. Start the event Heap that can be found in the Event Heap folder (./Event Heap /run.bat). This is the core of the system.
2. [Optional] Start the event logger (./Event Logger/run.bat). A very useful utility to inspection the events that are sent to the event bus.
3. Start the Patch Panel core (./Patch Panel/run.bat). The element that maintain the relations and mappings between events.

4. Start the Patch Panel Manager(./Patch Panel Manager/run.bat). This is the GUI for the patch panel that users can interact to create easily relations and mappings between events. Also very useful because it permits to users to load the rules that Patch Panel will follow, a more efficient way to program it.

If there appear any problem executing the previous steps, the files have to be recompiled manually or in some cases, if available, using the build.bat file.

After all parts are started, the program TerminalEventSender (./Software proxies/TerminalEventSender/run.bat) can be used to test the environment.

Terminal Event Sender is a software proxy that permits to send easily events to event heap throw console interface

The usage is simple:

```
run.bat <event heap name> <event type> {<field name:value>}*
```

where the event heap name is the name of the machine where Event Heap is running, “localhost” in the case of running the event Heap in the same computer.

2.2. WiiRemote

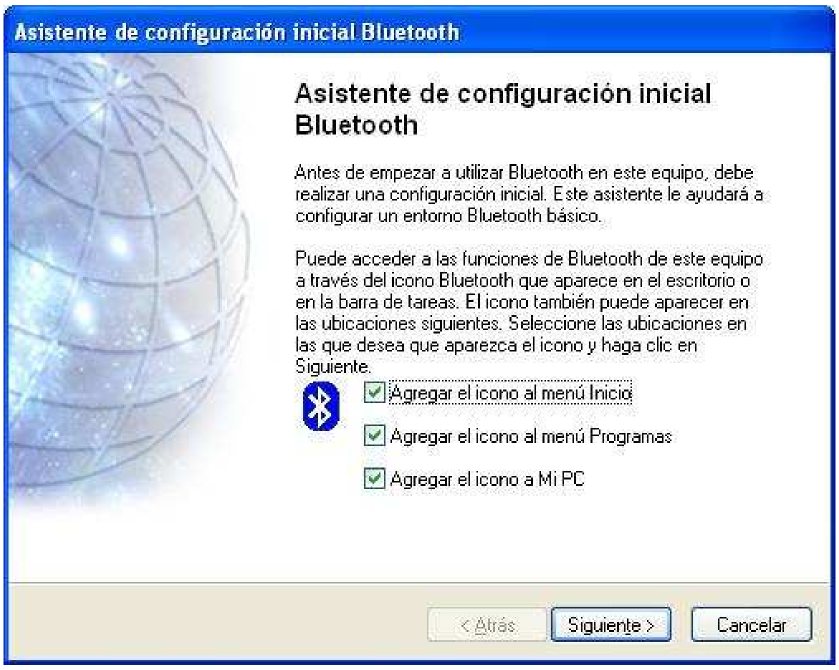
This section describes how to connect WiiRemote under Windows XP operating system. Note that a Bluetooth adapter is required.

In our tests, we have used the following USB device to set up a Bluetooth interface for our computer:

- ***Kensington Bluetooth USB Adapter 2.0*** [16]

We need to install a L2CAP supported Bluetooth stack. WIDECOMM drivers are selected because are the only one that support L2CAP under Windows OS environment [18]. The process to install WIDECOMM Bluetooth stack can be found in [17].

An important detail is that after the WIDECOMM stack is installed following the procedure described previously; do not use the Windows default Bluetooth wizard (Fig. 10) because the Bluetooth stack stop working for our java environment.



Asistente de configuración inicial Bluetooth

Asistente de configuración inicial Bluetooth

Antes de empezar a utilizar Bluetooth en este equipo, debe realizar una configuración inicial. Este asistente le ayudará a configurar un entorno Bluetooth básico.

Puede acceder a las funciones de Bluetooth de este equipo a través del icono Bluetooth que aparece en el escritorio o en la barra de tareas. El icono también puede aparecer en las ubicaciones siguientes. Seleccione las ubicaciones en las que desea que aparezca el icono y haga clic en Siguiente.

- [✓] Agregar el icono al menú Inicio
- [✓] Agregar el icono al menú Programas
- [✓] Agregar el icono a Mi PC

Fig. 10 Windows Bluetooth Wizard. Always click on Cancel.

The Windows XP Bluetooth wizard can be found on the System task bar as the following figure show (Fig. 11).

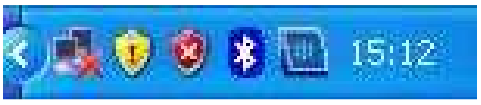


Fig. 11 Bluetooth Wizard Icon on Windows XP

2.3. WiiGee java libraries

To start testing WiiGee Library we need the following software:

- WiiGeeDemo GUI [14]
- Java 5.0 or Later (in our case Java 6.0 is used)
- Java 3D [15]
- BlueCove as Bluetooth library (included in WiiGee Demo GUI)
- (optional) Eclipse v. 3.5.1 as software development kit.

Simply, after Java 6.0, Java 3D and Eclipse SDK are installed we have to start the Eclipse SDK and import the project where WiiGeeDemo GUI (Figure 12) is extracted from the packaged downloaded file.

To start working with the WiiGeeDemo first we need to add the following dependences of java3D into the project:

- j3dcore.jar
- j3dutils.jar
- vecmath.jar

Those libraries can be found in Java3d installation path (in our case, C:\Archivos de programa\Java\Java3D\1.5.2\lib\ext).

Add the WiiGee libraries that can be found in the WiiGee-demoGUI extracted directory and modify the image path of the loaded images on the WiimotePanel.java to adapt to Windows OS path.

Two lines have to be added in the main class (wiigeemain.java). The first one to avoid Bluetooth windows bug and the second one to detect by default the WIDECOMM Bluetooth stack by the application, otherwise, Windows environment will start with Winsock stack, which is not the appropriate because don't support L2CAP.

```
System.setProperty("bluecove.jsr82.psm_minimum_off", "true");
System.setProperty("bluecove.stack.first", "widcomm");
```

Now we can start to test the WiiGeeDemo GUI. To assure that is correct started the following messages have to appear on the Log Console:

```
This is wiigee version 1.5.6 (20090817)
This is wiigee-plugin-wiimote version 1.5.6 (20090817)
BlueCove version 2.1.0 on widcomm
You are using the widcomm Bluetooth stack (Version BWT 5.1.0.1100, DK
6.1.0.1502)
L2CAP supported: true
wiigee: found a supported stack!
loading canvas3d
```

In the previous log message we have to identify that L2CAP are supported.

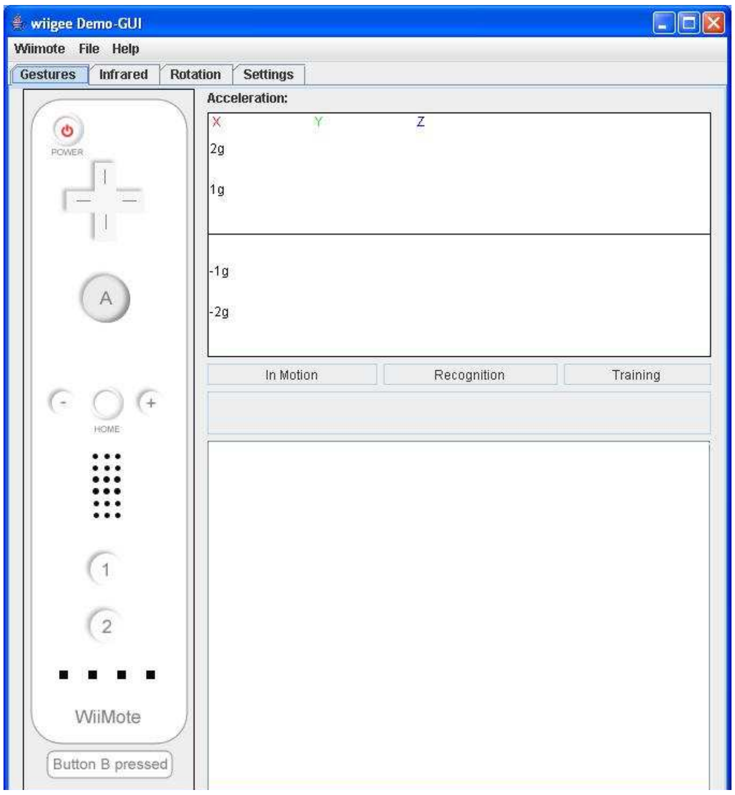


Fig. 12 WiiGee GUI

The first task is to attach the WiiRemote to the Bluetooth WIDECOMM stack. This can be done using the menu: WiiGee → Autoconnect. Notice that the user has to press simultaneously the WiiGee 1 & 2 buttons to be detected by our Bluetooth stack.

If the previous process worked successfully, on the log you can see the following message:

```
Starting device inquiry...
Device discovered: 0026596761AB -
Is a Wiimote!
Inquiry completed.
Device discovery completed!
WiimoteStreamer initialized...
WiimoteStreamer running...
Unknown data retrieved.
 A1 30 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Autocalibration successful!
Enabling ACCELEROMETER...
Enabling ACCELEROMETER...
```

And in the GUI if you start moving the WiiRemote you can see something similar to the following figure:

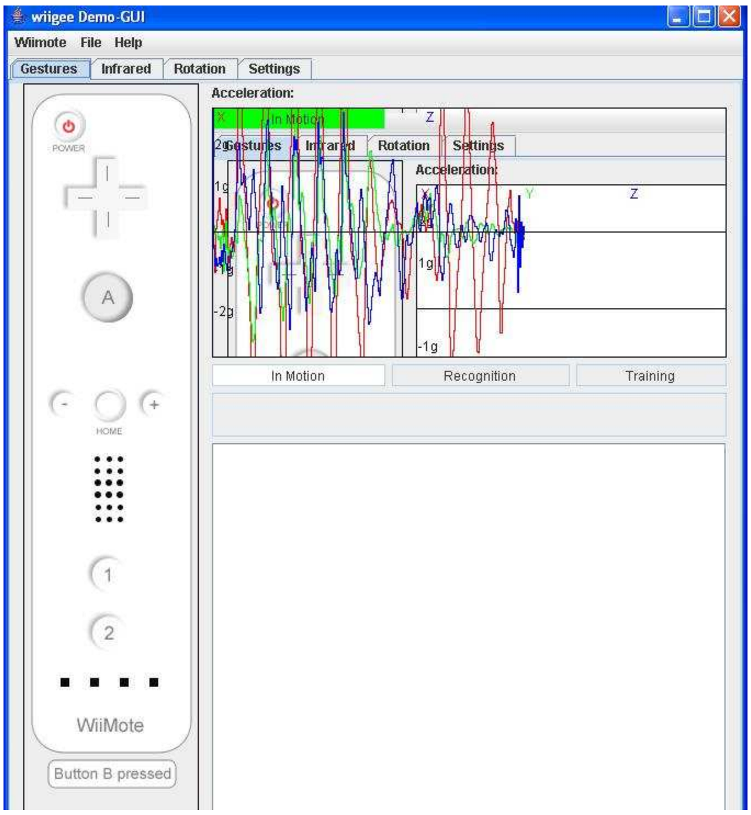


Fig. 13 WiiGee GUI successfully start up.

By default, “A” button in WiiRemote start the Training Process and B button the recognition process. Once you have a gesture trained several times (recommended by library developers to do between 5 and 10 trains) you can use the HOME button to assign a name to that trained gesture, save it and continue training new gestures using the same process.

After some gestures are trained, to facilitate the startup and avoid the training process of the gestures on each start up of the wiiGee program, a gesture set can be saved using the options “Save Gestureset” (File → Save Gestureset). Now each time we need to load the previous trained gestures, we can use the “load Gestureset” option (available on menu File → Load Gestureset).

Note that when we save a gesture a .txt document with the available information of the gesture is saved on hard disk and when a gesture set is saved a .wgs file is created that contains the name of the files (avoiding the extension) that contains each gesture individually. Obviously, the gesture set (file with .wgs extension) is which we need to load into the WiiGee GUI.

2.4. Test scenario

This section describes the scenarios created to test firstly iStuff library first alone and then used together with wiiGee libraries and a wiiMote. The final test is shown as example of interaction between WiiRemote, attached to wiiGee libraries. Those components use an iStuff proxy to permit the communication along the scenario with the collaborative application.

The general objectives of those tests are:

- Determine capabilities of wiiGee library for detecting different gestures and capabilities of iStuff architecture to create an ubiquitous test environment.
- Creation and integration of an iStuff proxy in wiiGeeGUI.
- Create a ubiquitous test environment in which all the parts can interact together.

2.4.1 iStuff library

The objective of this scenario is to test the correct running of the iStuff architecture by creating a basic Patch Panel rule and send manual events to the event heap using the Terminal Event Sender program.

To test the iStuff library capabilities the following scenario represented in Fig. 14 is mounted:

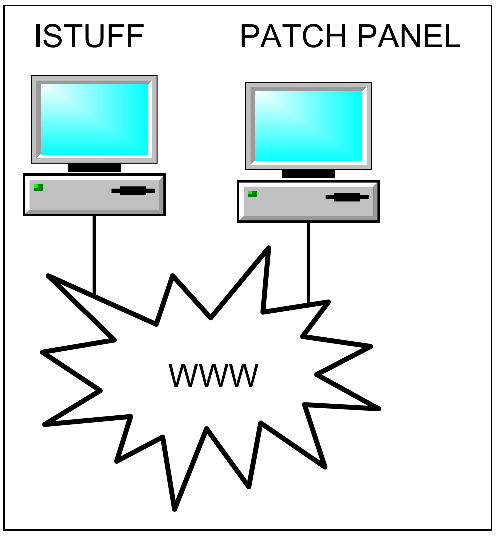


Fig. 14 iStuff alone test scenario

The first scenario is composed by two PCs, “Istuff” and “Patch Panel” with the components:

- **ISTUFF:**
    - Hostname: istuff
    - IP: 147.83.118.138
    - OS: Windows XP
    - ISTUFF components: Event heap and Event Logger

- **PATCH PANEL:**
    - Hostname: dac-epsc-1
    - IP: 147.83.118.125
    - OS: Windows XP
    - ISTUFF components: Patch Panel and the software proxy Terminal Event Sender.

Using Event Logger the events that are sent to the event heap can be visualized.

The rules that are programmed on the patch panel are the following:

```
program SimpleDemo;

event saluda type hello {
}

event saludat type helloHi{
}
initial state Initial {
 on saluda{
 send saludat;
}
}
```

Those simple rules on a received packet type “hello” send an empty packet type “helloHi”.

The modified code of the TerminalEventSender that is used on this test can be found on Annex 1.

2.4.2 WiiGee library and iStuff library

The objective of this section is connecting wiiGee libraries throw a proxy with the iStuff event bus.

Fist we need to attach the WiiMote to the WiiGee library as is described in section 2.6. Once is done, a proxy to connect wiiGee to iStuff bus is required.

We have based iStuff proxy on the TerminalEventSender example that iStuff library includes. The code of the created class (called iStuffProxy.java) can be found on Annex 2. Remember to add manually iStuff libraries to WiiMote project for success working. This class connects to iStuff bus on a host called “istuff” and creates a packet of type WiiEvent with a field called “WiiComand” that contains the name of the gesture recognized by WiiGee library.

After this, we have to add our proxy into the wiiGee source code. The easiest way to do it consist in adding a call to “arrenca” method with the name of the recognized event as a parameter. To do this, gestureRecieved method (can be found on FrontEnd.java of WiiGee GUI program found on WiiGee libraries) is modified as follows:

```
public void gestureReceived(GestureEvent event) {
     IStuffProxy ISP = new IStuffProxy();

        if(event.isValid()) {
            this.gestureField.setBackground(Color.GREEN);
            this.gestureField.setForeground(Color.WHITE);
            this.gestureField.setText("Gesture " +
this.gestureMeanings.elementAt(event.getId()) + " received.");
            //enviar el evento a la xarxa
            ISP.arrenca(this.gestureMeanings.elementAt(event.getId()));

            this.appendToConsole("Gesture
"+this.gestureMeanings.elementAt(event.getId())+" received.");
        } else {
            this.gestureField.setBackground(Color.RED);
            this.gestureField.setForeground(Color.WHITE);
            this.gestureField.setText("No Gesture recognized!");
            this.appendToConsole("No Gesture recognized!");
        }
    }
```

Highlighted lines indicate the code added. We can observe that first a new instance of the IStuffProxy is created and then the recognized event name is transferred to the created iStuffProxy to be sent as a parameter of the field “WiiComand” into a packet of iStuff architecture.

The second step consists on creating the rules for iStuff patch panel as follows:

```
program Demo;

event WiiUp type WiiEvent {
   string WiiCommand = "amunt";
}
event WiiDown type WiiEvent {
   string WiiCommand = "avall";
}
event WiiLeft type WiiEvent {
   string WiiCommand = "esquerra";
}
event WiiRight type WiiEvent {
   string WiiCommand = "dreta";
}
event WiiShake type WiiEvent {
   string WiiCommand = "shake";
}
event WiiCorbe type WiiEvent {
   string WiiCommand = "corbe";
}
event ControlUp type ControlEvent {
   string ControlCommand = "Up";
}
event ControlDown type ControlEvent {
   string ControlCommand = "Down";
}
event ControlLeft type ControlEvent {
   string ControlCommand = "Left";
}
event ControlRight type ControlEvent {
   string ControlCommand = "Right";
}
event ControlWindows type ControlEvent {
   string ControlCommand = "Windows";
}
event ControlExecute type ControlEvent {
   string ControlCommand = "Execute";
}



initial state Initial {
   on WiiUp {
        send ControlUp;
   }
        on WiiDown {
        send ControlDown;
   }
```

```
        on WiiLeft {
        send ControlLeft;
   }
        on WiiRight {
        send ControlRight;
   }
        on WiiShake {
        send ControlWindows;
   }
        on WiiCorbe {
        send ControlExecute;
   }

}
```

In this script we define the alias for received events and for the event to be sent into bus. You can distinguish between types “WiiEvent” and “ControlEvent”:

- WiiEvent type is related to the packets sent by the iStuff proxy attached to the wiiGee component (which recognizes wiiRemote movements).

- ControlEvent is which will be received by another iStuff proxy described in the following lines and is generated by the patch panel and as a consequence of receiving a WiiEvent packet.

We also need to determine the rules that patch panel are following: after “initial state Initial” of the previous example rules are shown. For example on a WiiDown received event, patch panel will send a ControlDown event and on a WiiCorbe received event, patch panel will send a ControlWindows event as we defined.

To complete the scenario, another proxy attached to the bus is created to test the success working. In this case, the objective is to generate some simple events to interact with Windows operating system.

The events that are selected to emulate the input keyboard are:
- Move up button
- Move down button
- Move left button
- Move right button
- Windows button
- Intro button

The program created called “ControlledWindows” (source code of ControlledWindows.java on Annex 2) uses the robot java class to emulate the keyboard pressed keys.

Finally the scenario to mount as is shown in the following Fig. 15:

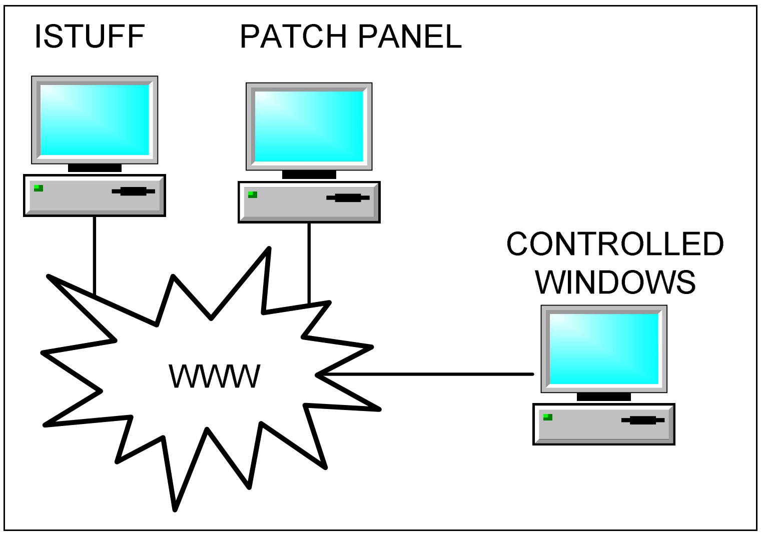


Fig. 15 Scenario 2: Controlled Windows

This second scenario is formed by three machines with the following characteristics:

- **ISTUFF :**
    - Hostname: istuff
    - IP: 147.83.118.126
    - ISTUFF components: Event heap and Event Logger
    - WiiGee GUI with integrated iStuff proxy.

- **PATCH PANEL:**
    - Hostname: dac-epsc-1
    - IP: 147.83.118.125
    - ISTUFF components: Patch Panel.

- **CONTROLLED WINDOWS:**
    - Hostname: jose-pc
    - IP: 147.83.118.127
    - Controlled windows application.

In the next chapter (section 3.2) you can observe the results of this test scenario.

2.4.3 Collaborative application

There exists a close relation between ubiquitous computer and new human-to-computer interfaces. The purpose of this section is to create a demonstration of how new human-to-computer interface, with Wii Remote device as example, can interact into a ubiquitous computing environment, iStuff event bus, and using a collaborative application.

The collaborative application created consists in a modified Tetris adapted for two players to play simultaneously the same board. This Tetris but uses as input the generated Wii Remote events sent by WiiGee GUI that integrates an iStuff proxy (as described on the previous point). The objective of the game is the same as classical Tetris, do the maximum number of point completing full lines without spaces; but in this case, you have to cooperate with the other player to move correctly the pieces on the board to complete the objective.

As in the previous scenario, the first step consists in connecting the wiiGee GUI to the event Heap using a proxy. The same modified class is used (FrontEnd.java) to communicate WiiGee GUI and iStuff event heap.

The next step consists in creating the rules that iStuff patch panel will follow to map the events generated by the WiiGee GUI proxy to events that are understandable by the Tetris iStuff proxy also connected to the iStuff event Heap.

The rules that are loaded on patch panel are the following:

```
program Demo;

event WiiUp type WiiEvent {
        string SourceDevice = "ISTUFF";
        string WiiCommand = "amunt";

}
event WiiDown type WiiEvent {
        string SourceDevice = "ISTUFF";
        string WiiCommand = "avall";

}
event WiiLeft type WiiEvent {
        string SourceDevice = "ISTUFF";
        string WiiCommand = "esquerra";
}
event WiiRight type WiiEvent {
        string SourceDevice = "ISTUFF";
        string WiiCommand = "dreta";

}
event ControlUp type ControlEvent {
   string ControlCommand = "Up";
}
event ControlDown type ControlEvent {
   string ControlCommand = "Down";
}
event ControlLeft type ControlEvent {
   string ControlCommand = "Left";
}
event ControlRight type ControlEvent {
   string ControlCommand = "Right";
}
```

```
event WiiUp2 type WiiEvent {
          string SourceDevice = "espc-1";
          string WiiCommand = "amunt";
}
event WiiDown2 type WiiEvent {
          string SourceDevice = "espc-1";
          string WiiCommand = "avall";
}
event WiiLeft2 type WiiEvent {
          string SourceDevice = "espc-1";
          string WiiCommand = "esquerra";
}
event WiiRight2 type WiiEvent {
          string SourceDevice = "dac-espc-1";
          string WiiCommand = "dreta";
}
event ControlUp2 type ControlEvent {
   string ControlCommand = "Up2";
}
event ControlDown2 type ControlEvent {
   string ControlCommand = "Down2";
}
event ControlLeft2 type ControlEvent {
   string ControlCommand = "Left2";
}
event ControlRight2 type ControlEvent {
   string ControlCommand = "Right2";
}
initial state Initial {
   on WiiUp {
          send ControlUp;
   }
          on WiiDown {
          send ControlDown;
   }
          on WiiLeft {
          send ControlLeft;
   }
          on WiiRight {
          send ControlRight;
   }
          on WiiUp2 {
          send ControlUp2;
   }
          on WiiDown2 {
          send ControlDown2;
   }
          on WiiLeft2 {
          send ControlLeft2;
   }
          on WiiRight2 {
          send ControlRight2;
   }

}
```

In this case, only the normal movements of keyboard are adapted for both players:

- Move up button
- Move down button
- Move left button
- Move right button

To distinguish between player one and player two input events we have used the "SourceDevice" (notice that iStuff software distinguish between majuscule and minuscule letters) field that is always included in every event send to event head. In this scenario player one is using the WiiMote that is connected to "ISTUFF" host and player two the WiiMote connected to "espc-1" host.

Finally, and as last step, we need to create the collaborative application:
A Tetris application that can be found on http://www.percederberg.net is used (http://www.percederberg.net/software/tetris/tetris-1.2-src.zip) as starting point.

This application is based on the following classes:

- Configuration.java: Provides the static methods for simplifying the reading of configuration parameters. It also includes some methods for transforming string values into more useful objects.
- Figure.java: Represents Tetris figures. Each one consists of four connected squares on seven possible constellations. Each figure can have to states; attached to a square or not. When is attached, no rotation or movements of pieces are allowed; when is not, a rotation can be made.
- Game.java: Controls all events in the game and handles all the game logics. Also implements the graphical game component.
- SquareBoard.java: Tetris square board. Rectangular board that contains a grid of colored squares representing the pieces.
- Main.java: Program main class. Set up the frames, listeners and starts the game

To adapt the obtained code for our purpose, we modified the Game.java class to instead of one piece, two pieces are created (source code can be found on Annex 3, Game.java). We also need to modify all the game handlers to complete the code.

Also keyboard keys are assigned to permit the movement of both players' pieces into the square board.

At this point, we have a Tetris game for two players but is still an Istuff proxy required. We have created an Istuff proxy class, called IstuffProxy.java (source code is available on Annex 3).

The integration of IStuffProxy and the Game is done using an EventListener Java object that is implemented by the IStuffProxy class. Then a method that listens to these events is created on Game.java (called onReceivedSignal) that implements the right movement of the piece according to the received message from IStuff event heap.

The following Fig. 16 shows the schematic of the proposed test scenario:

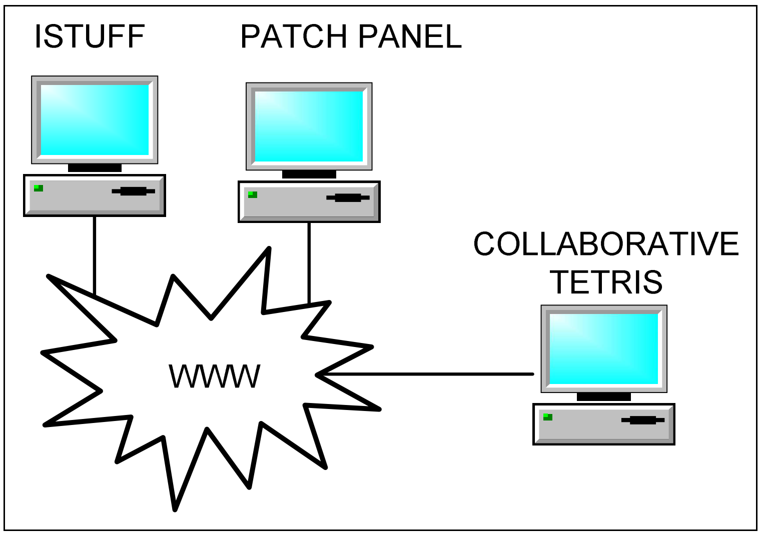


Fig. 16 Collaborative Tetris test scenario.

The scenario is composed by three hosts with the following specs:

- **ISTUFF :**
  - Hostname: istuff
  - IP: 147.83.118.126
  - ISTUFF components: Event heap and Event Logger
  - WiiGeeGui with integrated iStuff proxy. Acts as player 1 in the collaborative Tetris.

- **PATCH PANEL:**
  - Hostname: espc-1
  - IP: 147.83.118.124
  - ISTUFF components: Patch Panel.
  - WiiGeeGUI with integrated iStuff proxy. Acts as player 2 in the collaborative Tetris.

- **COLLABORATIVE TETRIS:**
  - Hostname: jose-pc
  - IP: 147.83.118.199
  - Collaborative Tetris application.

In the following section 3.3 you can find the results of this test scenario.

CHAPTER 3. Tests

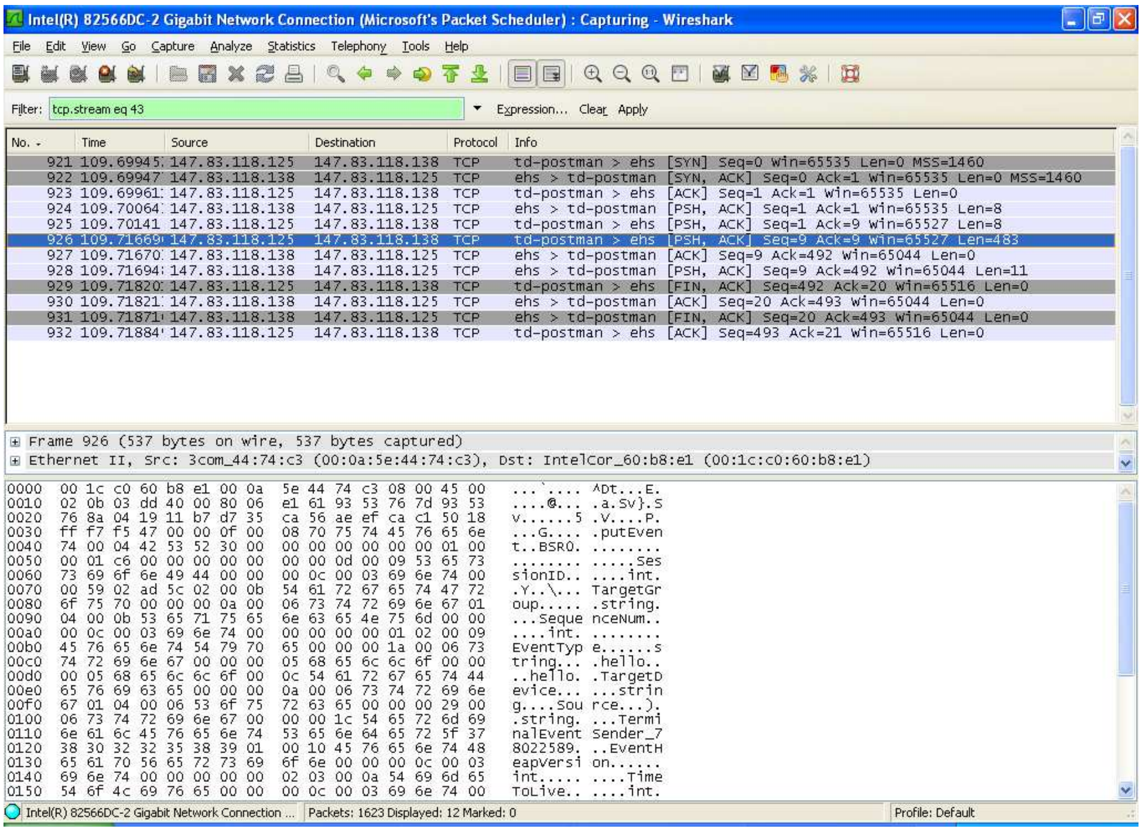
This chapter summarizes the results obtained from the tests that are described on chapter 2 in section 2.4.

3.1. iStuff test results

Remember in this first scenario, a PC called “Istuff” with the istuff components “Event Heap” and “Event Logger”, and in the second pc called “dac-epsc-1” the patch panel (patch panel and patch panel GUI) and the Terminal event sender are running.

To illustrate how istuff event bus (event Heap) works, in first test only the event heap, event logger and the Terminal Event Sender are running.

In this case we can observe the following traffic in the net (figure 17):


Intel(R) 82566DC-2 Gigabit Network Connection (Microsoft's Packet Scheduler) : Capturing - Wireshark
File Edit View Go Capture Analyze Statistics Telephony Tools Help
Filter: tcp.stream eq 43
Expression... Clear Apply

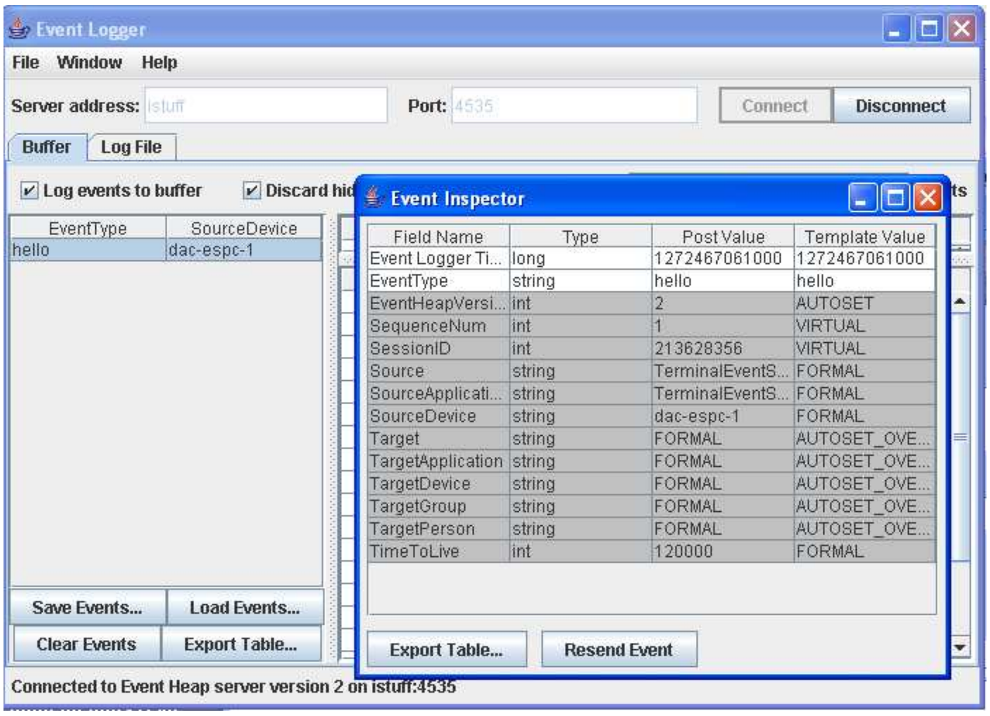
```
No. -  Time       Source          Destination     Protocol Info
   921 109.69945. 147.83.118.125  147.83.118.138  TCP      td-postman > ehs [SYN] Seq=0 Win=65535 Len=0 MSS=1460
   922 109.69947  147.83.118.138  147.83.118.125  TCP      ehs > td-postman [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
   923 109.69961: 147.83.118.125  147.83.118.138  TCP      td-postman > ehs [ACK] Seq=1 Ack=1 Win=65535 Len=0
   924 109.70064: 147.83.118.138  147.83.118.125  TCP      ehs > td-postman [PSH, ACK] Seq=1 Ack=1 Win=65535 Len=8
   925 109.70141  147.83.118.125  147.83.118.138  TCP      td-postman > ehs [PSH, ACK] Seq=1 Ack=9 Win=65527 Len=8
   926 109.71669' 147.83.118.125  147.83.118.138  TCP      td-postman > ehs [PSH, ACK] Seq=9 Ack=9 Win=65527 Len=483
   927 109.71670: 147.83.118.138  147.83.118.125  TCP      ehs > td-postman [ACK] Seq=9 Ack=492 Win=65044 Len=0
   928 109.71694: 147.83.118.138  147.83.118.125  TCP      ehs > td-postman [PSH, ACK] Seq=9 Ack=492 Win=65044 Len=11
   929 109.71820: 147.83.118.125  147.83.118.138  TCP      td-postman > ehs [FIN, ACK] Seq=492 Ack=20 Win=65516 Len=0
   930 109.71821: 147.83.118.138  147.83.118.125  TCP      ehs > td-postman [ACK] Seq=20 Ack=493 Win=65044 Len=0
   931 109.71871' 147.83.118.138  147.83.118.125  TCP      ehs > td-postman [FIN, ACK] Seq=20 Ack=493 Win=65044 Len=0
   932 109.71884' 147.83.118.125  147.83.118.138  TCP      td-postman > ehs [ACK] Seq=493 Ack=21 Win=65516 Len=0

⊞ Frame 926 (537 bytes on wire, 537 bytes captured)
⊞ Ethernet II, Src: 3com_44:74:c3 (00:0a:5e:44:74:c3), Dst: IntelCor_60:b8:e1 (00:1c:c0:60:b8:e1)

0000  00 1c c0 60 b8 e1 00 0a  5e 44 74 c3 08 00 45 00   ...`.... ^Dt...E.
0010  02 0b 03 dd 40 00 80 06  e1 61 93 53 76 7d 93 53   ....@... .a.Sv}.S
0020  76 8a 04 19 11 b7 d7 35  ca 56 ae ef ca c1 50 18   v......5 .V....P.
0030  ff f7 f5 47 00 00 0f 00  08 70 75 74 45 76 65 6e   ...G.... .putEven
0040  74 00 04 42 53 52 30 00  00 00 00 00 00 00 01 00   t..BSR0. ........
0050  00 01 c6 00 00 00 00 00  00 00 0d 00 09 53 65 73   ........ .....Ses
0060  73 69 6f 6e 49 44 00 00  00 0c 00 03 69 6e 74 00   sionID.. ....int.
0070  00 59 02 ad 5c 02 00 0b  54 61 72 67 65 74 47 72   .Y..\... TargetGr
0080  6f 75 70 00 00 00 0a 00  06 73 74 72 69 6e 67 01   oup..... .string.
0090  04 00 0b 53 65 71 75 65  6e 63 65 4e 75 6d 00 00   ...Seque nceNum..
00a0  00 0c 00 03 69 6e 74 00  00 00 00 00 01 02 00 09   ....int. ........
00b0  45 76 65 6e 74 54 79 70  65 00 00 00 1a 00 06 73   EventTyp e......s
00c0  74 72 69 6e 67 00 00 00  05 68 65 6c 6c 6f 00 00   tring... .hello..
00d0  00 05 68 65 6c 6c 6f 00  0c 54 61 72 67 65 74 44   ..hello. .TargetD
00e0  65 76 69 63 65 00 00 00  0a 00 06 73 74 72 69 6e   evice... ...strin
00f0  67 01 04 00 06 53 6f 75  72 63 65 00 00 00 29 00   g....Sou rce...).
0100  06 73 74 72 69 6e 67 00  00 00 1c 54 65 72 6d 69   .string. ...Termi
0110  6e 61 6c 45 76 65 6e 74  53 65 6e 64 65 72 5f 37   nalEvent Sender_7
0120  38 30 32 32 35 38 39 01  00 10 45 76 65 6e 74 48   8022589. ..EventH
0130  65 61 70 56 65 72 73 69  6f 6e 00 00 00 0c 00 03   eapVersi on......
0140  69 6e 74 00 00 00 00 00  02 03 00 0a 54 69 6d 65   int..... ....Time
0150  54 6f 4c 69 76 65 00 00  00 0c 00 03 69 6e 74 00   ToLive.. ....int.
```

Intel(R) 82566DC-2 Gigabit Network Connection ...
Packets: 1623 Displayed: 12 Marked: 0
Profile: Default

Fig. 17 Wireshark capture of TCP connection involved on test

Initially, a TCP connection between the two hosts is done. First the “dac-espc-1” host registers to the event Heap and sends the message. After this it deregisters from event heap and TCP connection terminates.

Event logger application captured the only packet that is send to the event heap (Fig. 18):



Server address: istuff
Port: 4535

☑ Log events to buffer ☑ Discard hid

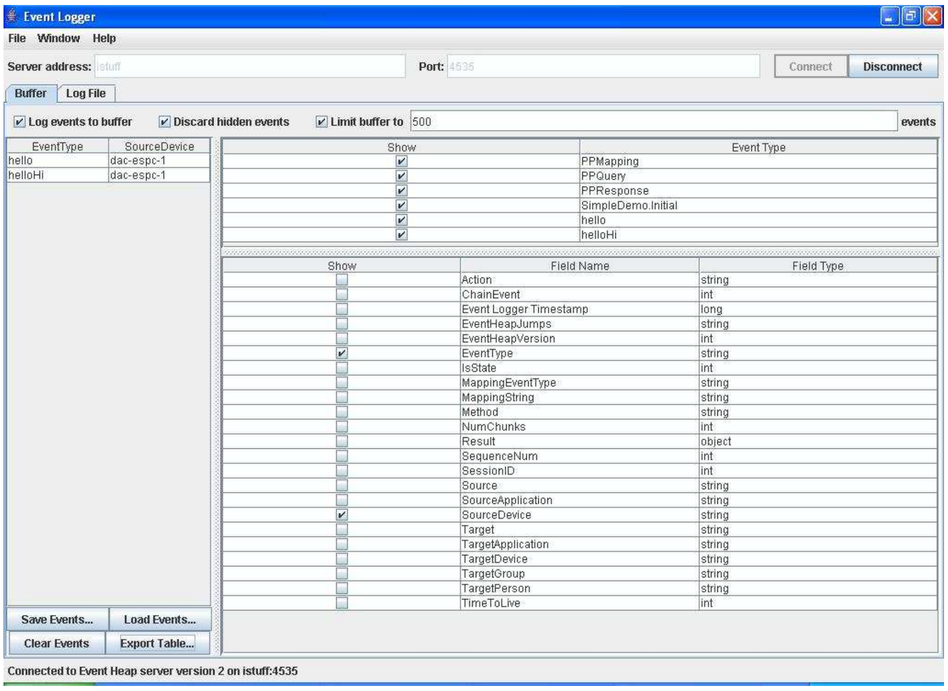
EventType	SourceDevice
hello	dac-espc-1

Field Name	Type	Post Value	Template Value
Event Logger Ti...	long	1272467061000	1272467061000
EventType	string	hello	hello
EventHeapVersi...	int	2	AUTOSET
SequenceNum	int	1	VIRTUAL
SessionID	int	213628356	VIRTUAL
Source	string	TerminalEventS...	FORMAL
SourceApplicati...	string	TerminalEventS...	FORMAL
SourceDevice	string	dac-espc-1	FORMAL
Target	string	FORMAL	AUTOSET_OVE...
TargetApplication	string	FORMAL	AUTOSET_OVE...
TargetDevice	string	FORMAL	AUTOSET_OVE...
TargetGroup	string	FORMAL	AUTOSET_OVE...
TargetPerson	string	FORMAL	AUTOSET_OVE...
TimeToLive	int	120000	FORMAL

Connected to Event Heap server version 2 on istuff:4535

Fig. 18 Hello EventType packet

Now, we start patch panel component (and the patch panel manager, which permits us to visualize programmed rules on patch panel). We load the rules prepared for this test, and start again the modified Terminal Event Sender and we can observe the following Fig. 19:



Server address: istuff
Port: 4535

☑ Log events to buffer ☑ Discard hidden events ☑ Limit buffer to 500 events

EventType	SourceDevice
hello	dac-espc-1
helloHi	dac-espc-1

Show	Event Type
☑	PPMapping
☑	PPQuery
☑	PPResponse
☑	SimpleDemo.Initial
☑	hello
☑	helloHi

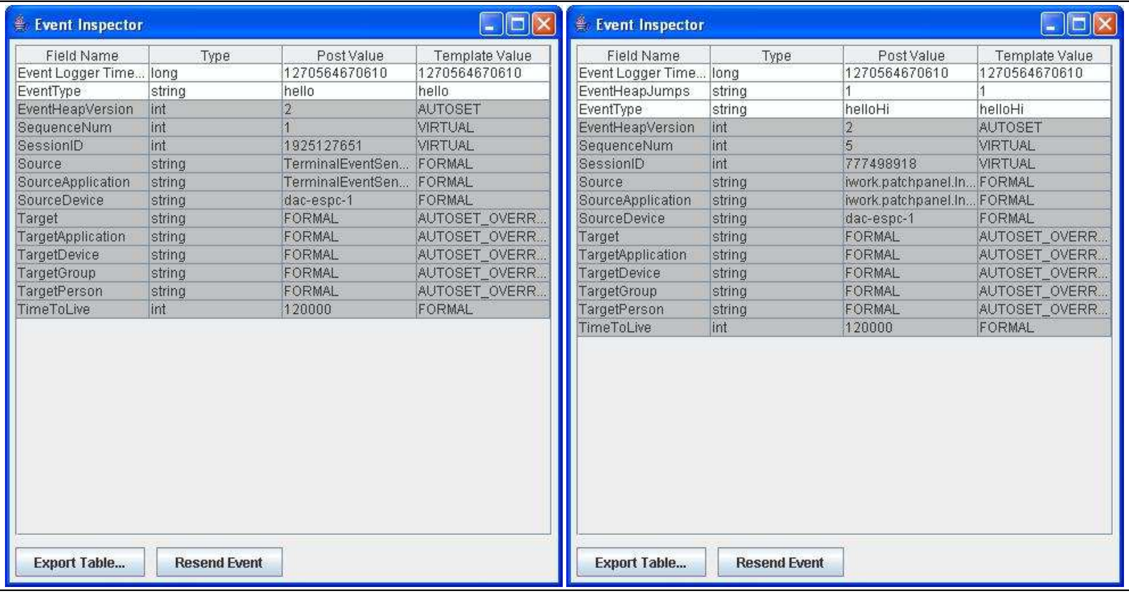
Show	Field Name	Field Type
☐	Action	string
☐	ChainEvent	int
☐	Event Logger Timestamp	long
☐	EventHeapJumps	string
☐	EventHeapVersion	int
☑	EventType	string
☐	IsState	int
☐	MappingEventType	string
☐	MappingString	string
☐	Method	string
☐	NumChunks	int
☐	Result	object
☐	SequenceNum	int
☐	SessionID	int
☐	Source	string
☐	SourceApplication	string
☑	SourceDevice	string
☐	Target	string
☐	TargetApplication	string
☐	TargetDevice	string
☐	TargetGroup	string
☐	TargetPerson	string
☐	TimeToLive	int

Connected to Event Heap server version 2 on istuff:4535

Fig. 19 Event logger capture

On the first packet sent, (EventType:hello) the second packet is automatically generated by the patch panel (EventType:helloHi) as we requested with the rule shown on the previous chapter. In this case the source device of both packets is the same because Terminal Event Sender and Patch panel are running on the same host.

If we observe in detail we can appreciate that other values are send (Fig. 20):



Field Name	Type	Post Value	Template Value
Event Logger Time...	long	1270564670610	1270564670610
EventType	string	hello	hello
EventHeapVersion	int	2	AUTOSET
SequenceNum	int	1	VIRTUAL
SessionID	int	1925127651	VIRTUAL
Source	string	TerminalEventSen...	FORMAL
SourceApplication	string	TerminalEventSen...	FORMAL
SourceDevice	string	dac-espc-1	FORMAL
Target	string	FORMAL	AUTOSET_OVERR...
TargetApplication	string	FORMAL	AUTOSET_OVERR...
TargetDevice	string	FORMAL	AUTOSET_OVERR...
TargetGroup	string	FORMAL	AUTOSET_OVERR...
TargetPerson	string	FORMAL	AUTOSET_OVERR...
TimeToLive	int	120000	FORMAL

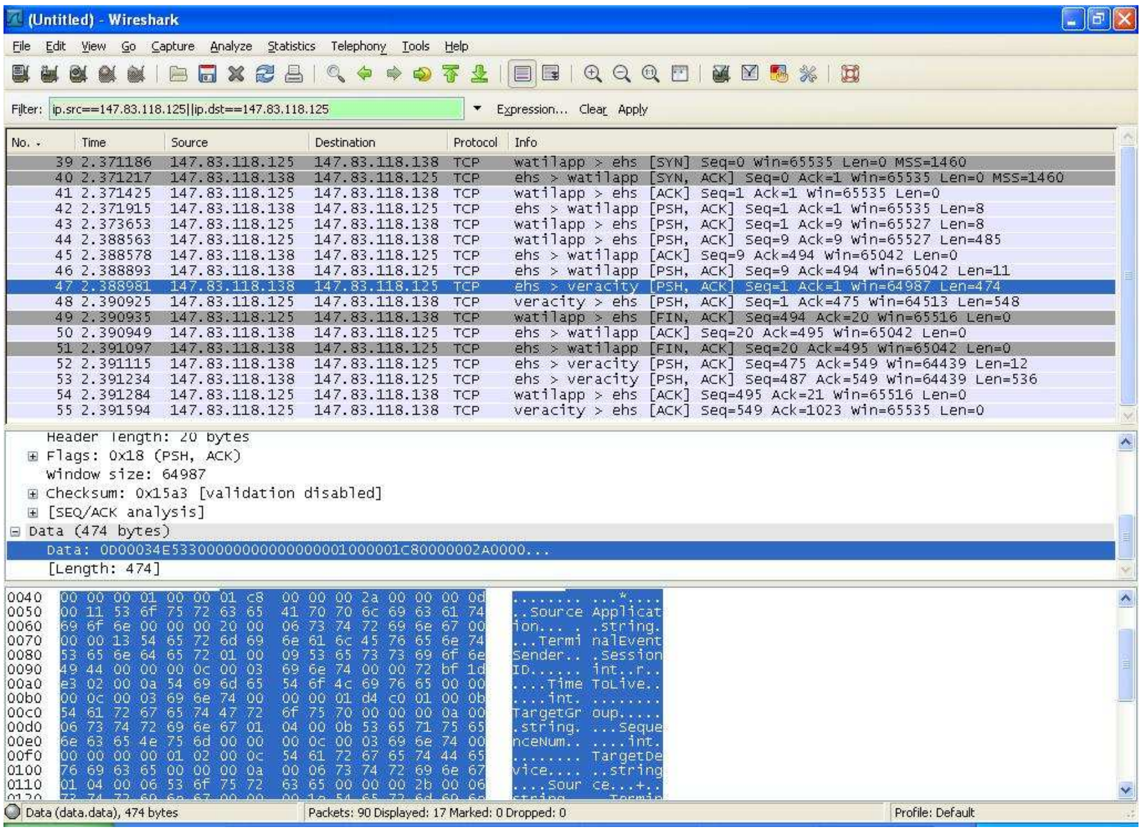
Field Name	Type	Post Value	Template Value
Event Logger Time...	long	1270564670610	1270564670610
EventHeapJumps	string	1	1
EventType	string	helloHi	helloHi
EventHeapVersion	int	2	AUTOSET
SequenceNum	int	5	VIRTUAL
SessionID	int	777498918	VIRTUAL
Source	string	iwork.patchpanel.In...	FORMAL
SourceApplication	string	iwork.patchpanel.In...	FORMAL
SourceDevice	string	dac-espc-1	FORMAL
Target	string	FORMAL	AUTOSET_OVERR...
TargetApplication	string	FORMAL	AUTOSET_OVERR...
TargetDevice	string	FORMAL	AUTOSET_OVERR...
TargetGroup	string	FORMAL	AUTOSET_OVERR...
TargetPerson	string	FORMAL	AUTOSET_OVERR...
TimeToLive	int	120000	FORMAL

Fig. 20 Hello & HelloHi packets in detail

There are other values sent by each element connected to the Event Heap. We can emphasize:

- Source Application: Identifies the software origin of the packet. In the first one we can observer that this origin corresponds to TerminalEventSender, and in the second one this is generated by iwork.patchpannel.Intermediary
- SourceDevice: Identifies the hardware origin of the packet. In both cases we can distinguish dac-epsc-1 (it's the name of the machine where PatchPannel and TerminalEventSender software are running).

In more detail, those packets are sent using TCP connection between both hosts as shown in figure 21:


(Untitled) - Wireshark
File Edit View Go Capture Analyze Statistics Telephony Tools Help
Filter: ip.src==147.83.118.125||ip.dst==147.83.118.125
Expression... Clear Apply
Profile: Default

```
No.   Time      Source          Destination     Protocol Info
   39 2.371186  147.83.118.125  147.83.118.125  TCP      watilapp > ehs [SYN] Seq=0 Win=65535 Len=0 MSS=1460
```

```
No.   Time      Source          Destination     Protocol Info
   39 2.371186  147.83.118.125  147.83.118.138  TCP      watilapp > ehs [SYN] Seq=0 Win=65535 Len=0 MSS=1460
   40 2.371217  147.83.118.138  147.83.118.125  TCP      ehs > watilapp [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
   41 2.371425  147.83.118.125  147.83.118.138  TCP      watilapp > ehs [ACK] Seq=1 Ack=1 Win=65535 Len=0
   42 2.371915  147.83.118.138  147.83.118.125  TCP      ehs > watilapp [PSH, ACK] Seq=1 Ack=1 Win=65535 Len=8
   43 2.373653  147.83.118.125  147.83.118.138  TCP      watilapp > ehs [PSH, ACK] Seq=1 Ack=9 Win=65527 Len=8
   44 2.388563  147.83.118.125  147.83.118.138  TCP      watilapp > ehs [PSH, ACK] Seq=9 Ack=9 Win=65527 Len=485
   45 2.388578  147.83.118.138  147.83.118.125  TCP      ehs > watilapp [ACK] Seq=9 Ack=494 Win=65042 Len=0
   46 2.388893  147.83.118.138  147.83.118.125  TCP      ehs > watilapp [PSH, ACK] Seq=9 Ack=494 Win=65042 Len=11
   47 2.388981  147.83.118.138  147.83.118.125  TCP      ehs > veracity [PSH, ACK] Seq=1 Ack=1 Win=64987 Len=474
   48 2.390925  147.83.118.125  147.83.118.138  TCP      veracity > ehs [PSH, ACK] Seq=1 Ack=475 Win=64513 Len=548
   49 2.390935  147.83.118.125  147.83.118.138  TCP      watilapp > ehs [FIN, ACK] Seq=494 Ack=20 Win=65516 Len=0
   50 2.390949  147.83.118.138  147.83.118.125  TCP      ehs > watilapp [ACK] Seq=20 Ack=495 Win=65042 Len=0
   51 2.391097  147.83.118.138  147.83.118.125  TCP      ehs > watilapp [FIN, ACK] Seq=20 Ack=495 Win=65042 Len=0
   52 2.391115  147.83.118.138  147.83.118.125  TCP      ehs > veracity [PSH, ACK] Seq=475 Ack=549 Win=64439 Len=12
   53 2.391234  147.83.118.138  147.83.118.125  TCP      ehs > veracity [PSH, ACK] Seq=487 Ack=549 Win=64439 Len=536
   54 2.391284  147.83.118.125  147.83.118.138  TCP      watilapp > ehs [ACK] Seq=495 Ack=21 Win=65516 Len=0
   55 2.391594  147.83.118.125  147.83.118.138  TCP      veracity > ehs [ACK] Seq=549 Ack=1023 Win=65535 Len=0
```

```
    Header length: 20 bytes
  ⊞ Flags: 0x18 (PSH, ACK)
    Window size: 64987
  ⊞ Checksum: 0x15a3 [validation disabled]
  ⊞ [SEQ/ACK analysis]
⊟ Data (474 bytes)
    Data: 0D00034E533000000000000000001000001C80000002A0000...
    [Length: 474]
```

```
0040  00 00 00 01 00 00 01 c8  00 00 00 2a 00 00 00 0d   ........ ...*....
0050  00 11 53 6f 75 72 63 65  41 70 70 6c 69 63 61 74   ..Source Applicat
0060  69 6f 6e 00 00 00 20 00  06 73 74 72 69 6e 67 00   ion... . .string.
0070  00 00 13 54 65 72 6d 69  6e 61 6c 45 76 65 6e 74   ...Termi nalEvent
0080  53 65 6e 64 65 72 01 00  09 53 65 73 73 69 6f 6e   Sender.. .Session
0090  49 44 00 00 00 0c 00 03  69 6e 74 00 00 72 bf 1d   ID...... int..r..
00a0  e3 02 00 0a 54 69 6d 65  54 6f 4c 69 76 65 00 00   ....Time ToLive..
00b0  00 0c 00 03 69 6e 74 00  00 00 01 d4 c0 01 00 0b   ....int. ........
00c0  54 61 72 67 65 74 47 72  6f 75 70 00 00 00 0a 00   TargetGr oup.....
00d0  06 73 74 72 69 6e 67 01  04 00 0b 53 65 71 75 65   .string. ...Seque
00e0  6e 63 65 4e 75 6d 00 00  00 0c 00 03 69 6e 74 00   nceNum.. ....int.
00f0  00 00 00 00 01 02 00 0c  54 61 72 67 65 74 44 65   ........ TargetDe
0100  76 69 63 65 00 00 00 0a  00 06 73 74 72 69 6e 67   vice.... ..string
0110  01 04 00 06 53 6f 75 72  63 65 00 00 00 2b 00 06   ....Sour ce...+..
```

```
Data (data.data), 474 bytes    Packets: 90 Displayed: 17 Marked: 0 Dropped: 0
```

Fig. 21 Wireshark capture

We can observe that all the fields that are visualized in the previous captures, are sent into the data fields of TCP packets.

We can describe the flow in more detail:

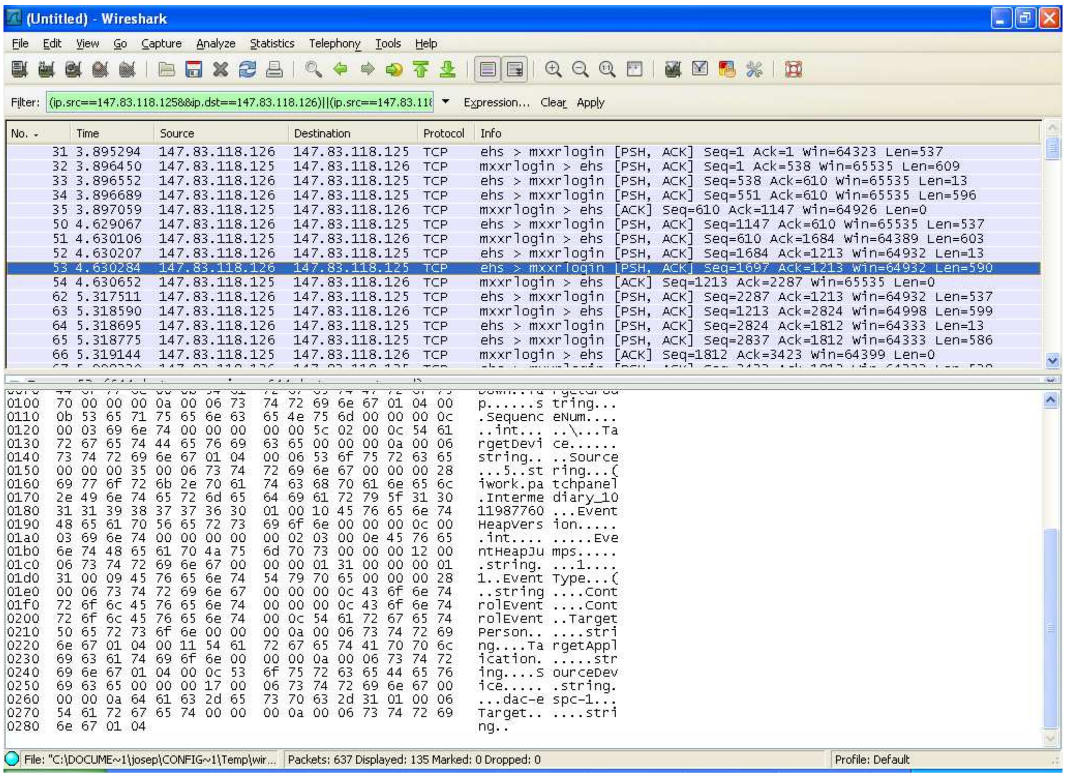
1. Terminal Event Sender registers to event Heap, sends the “Hello” packet and then deregisters. Terminal Event Sender program ends.
2. Event Heap receives the Hello packet and is forwarded to the registered patch panel.
3. Patch panel receives the Hello packet and looks up in its mapping. Finds out a rule that matches the received packet and forwards the corresponding packet, called “HelloHi”, to the event Heap.
4. Finally, event Heap forwards the “HelloHi” packet to all others hosts that are registered to event Heap.

3.2. WiiGee library and iStuff library test results

This scenario shows how Wiigee library (or the interaction done using a Wii remote device) is integrated with the iStuff library as a network communication path. To probe the correct working of the scenario also a proxy attached to the iStuff bus is created with the objective of control remotely an operating system. In this case, Windows Vista is the operating system chosen.

To start up this scenario we have to start first the event heap and event logger. Then we connect the patch panel to the event heap, start the patch panel manager in which we load the defined rules you can found on section 3.4.2 and finally we start the “controlledWindows” application.

The following Fig. 22 illustrates the dialog present in the bus between “ISTAFF” and “dac-espc-1” (event-heap and patch panel):


(Untitled) - Wireshark
File Edit View Go Capture Analyze Statistics Telephony Tools Help
Filter: (ip.src==147.83.118.125&&ip.dst==147.83.118.126)||(ip.src==147.83.11
Expression... Clear Apply
Profile: Default

```
No.  Time     Source         Destination    Protocol Info
  31 3.895294 147.83.118.126 147.83.118.125 TCP      ehs > mxxrlogin [PSH, ACK] Seq=1 Ack=1 Win=64323 Len=537
  32 3.896450 147.83.118.125 147.83.118.126 TCP      mxxrlogin > ehs [PSH, ACK] Seq=1 Ack=538 Win=65535 Len=609
  33 3.896552 147.83.118.126 147.83.118.125 TCP      ehs > mxxrlogin [PSH, ACK] Seq=538 Ack=610 Win=65535 Len=13
  34 3.896689 147.83.118.126 147.83.118.125 TCP      ehs > mxxrlogin [PSH, ACK] Seq=551 Ack=610 Win=65535 Len=596
  35 3.897059 147.83.118.125 147.83.118.126 TCP      mxxrlogin > ehs [ACK] Seq=610 Ack=1147 Win=64926 Len=0
  50 4.629067 147.83.118.126 147.83.118.125 TCP      ehs > mxxrlogin [PSH, ACK] Seq=1147 Ack=610 Win=65535 Len=537
  51 4.630106 147.83.118.125 147.83.118.126 TCP      mxxrlogin > ehs [PSH, ACK] Seq=610 Ack=1684 Win=64389 Len=603
  52 4.630207 147.83.118.126 147.83.118.125 TCP      ehs > mxxrlogin [PSH, ACK] Seq=1684 Ack=1213 Win=64932 Len=13
  53 4.630284 147.83.118.126 147.83.118.125 TCP      ehs > mxxrlogin [PSH, ACK] Seq=1697 Ack=1213 Win=64932 Len=590
  54 4.630652 147.83.118.125 147.83.118.126 TCP      mxxrlogin > ehs [ACK] Seq=1213 Ack=2287 Win=65535 Len=0
  62 5.317511 147.83.118.126 147.83.118.125 TCP      ehs > mxxrlogin [PSH, ACK] Seq=2287 Ack=1213 Win=64932 Len=537
  63 5.318590 147.83.118.125 147.83.118.126 TCP      mxxrlogin > ehs [PSH, ACK] Seq=1213 Ack=2824 Win=64998 Len=599
  64 5.318695 147.83.118.126 147.83.118.125 TCP      ehs > mxxrlogin [PSH, ACK] Seq=2824 Ack=1812 Win=64333 Len=13
  65 5.318775 147.83.118.126 147.83.118.125 TCP      ehs > mxxrlogin [PSH, ACK] Seq=2837 Ack=1812 Win=64333 Len=586
  66 5.319144 147.83.118.125 147.83.118.126 TCP      mxxrlogin > ehs [ACK] Seq=1812 Ack=3423 Win=64399 Len=0
```

```
0100  70 00 00 00 0a 00 06 73  74 72 69 6e 67 01 04 00   p......s tring...
0110  0b 53 65 71 75 65 6e 63  65 4e 75 6d 00 00 00 0c   .Sequenc eNum....
0120  00 03 69 6e 74 00 00 00  00 00 5c 02 00 0c 54 61   ..int... ..\...Ta
0130  72 67 65 74 44 65 76 69  63 65 00 00 00 0a 00 06   rgetDevi ce......
0140  73 74 72 69 6e 67 01 04  00 06 53 6f 75 72 63 65   string.. ..Source
0150  00 00 00 35 00 06 73 74  72 69 6e 67 00 00 00 28   ...5..st ring...(
0160  69 77 6f 72 6b 2e 70 61  74 63 68 70 61 6e 65 6c   iwork.pa tchpanel
0170  2e 49 6e 74 65 72 6d 65  64 69 61 72 79 5f 31 30   .Interme diary_10
0180  31 31 39 38 37 37 36 30  01 00 10 45 76 65 6e 74   11987760 ...Event
0190  48 65 61 70 56 65 72 73  69 6f 6e 00 00 00 0c 00   HeapVers ion.....
01a0  03 69 6e 74 00 00 00 00  00 02 03 00 0e 45 76 65   .int.... .....Eve
01b0  6e 74 48 65 61 70 4a 75  6d 70 73 00 00 00 12 00   ntHeapJu mps.....
01c0  06 73 74 72 69 6e 67 00  00 00 01 31 00 00 00 01   .string. ...1....
01d0  31 00 09 45 76 65 6e 74  54 79 70 65 00 00 00 28   1..Event Type...(
01e0  00 06 73 74 72 69 6e 67  00 00 00 0c 43 6f 6e 74   ..string ....Cont
01f0  72 6f 6c 45 76 65 6e 74  00 00 00 0c 43 6f 6e 74   rolEvent ....Cont
0200  72 6f 6c 45 76 65 6e 74  00 0c 54 61 72 67 65 74   rolEvent ..Target
0210  50 65 72 73 6f 6e 00 00  00 0a 00 06 73 74 72 69   Person.. ....stri
0220  6e 67 01 04 00 11 54 61  72 67 65 74 41 70 70 6c   ng....Ta rgetAppl
0230  69 63 61 74 69 6f 6e 00  00 00 0a 00 06 73 74 72   ication. .....str
0240  69 6e 67 01 04 00 0c 53  6f 75 72 63 65 44 65 76   ing....S ourceDev
0250  69 63 65 00 00 00 17 00  06 73 74 72 69 6e 67 00   ice..... .string.
0260  00 00 0a 64 61 63 2d 65  73 70 63 2d 31 01 00 06   ...dac-e spc-1...
0270  54 61 72 67 65 74 00 00  00 0a 00 06 73 74 72 69   Target.. ....stri
0280  6e 67 01 04                                        ng..
```

```
File: "C:\DOCUME~1\josep\CONFIG~1\Temp\wir...   Packets: 637 Displayed: 135 Marked: 0 Dropped: 0
```

Fig. 22 Bus activity during ControlledWindows test

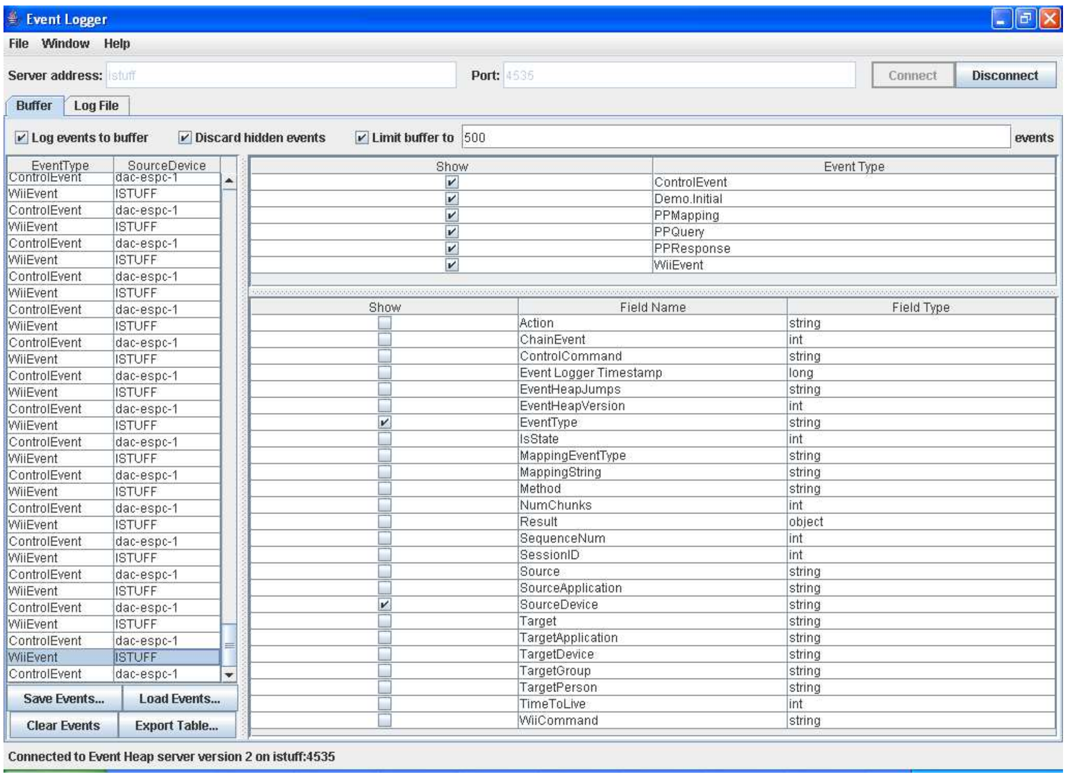
The process that is accomplished in that process is going to be described in detail:

1. Start event Heap bus.
2. Start event Logger and Event inspector: Both register in even Heap using a TCP connection that remains open.
3. Start Patch Panel and Patch Panel manager: Also both register to even Heap using a TCP connection that remains open.

4. Start WiiGeeGUI: Registers to even Heap with a TCP connection. Connection remains open.
5. Start ControlledWindows: It registers to the even Heap with a TCP connection that remains open.

At this point, when an event is sent to the Event Heap bus using the WiiGeeGUI istuff proxy, we can observe:

1. The event is spread out to other listeners as loggers (event Logger and Inspector) and to the patch panel.
2. Patch panel receives the event and looks up in rules. If event mapping exist, acts in agreement with predefined rules. In this case, another event is sent to the bus. The following figures (Fig.23 and 24) illustrates that fact:



Server address: istuff Port: 4535

☑ Log events to buffer ☑ Discard hidden events ☑ Limit buffer to 500 events

EventType	SourceDevice
ControlEvent	dac-espc-1
WiiEvent	ISTUFF
ControlEvent	dac-espc-1
WiiEvent	ISTUFF
ControlEvent	dac-espc-1
WiiEvent	ISTUFF
ControlEvent	dac-espc-1
WiiEvent	ISTUFF
ControlEvent	dac-espc-1
WiiEvent	ISTUFF
ControlEvent	dac-espc-1
WiiEvent	ISTUFF
ControlEvent	dac-espc-1
WiiEvent	ISTUFF
ControlEvent	dac-espc-1
WiiEvent	ISTUFF
ControlEvent	dac-espc-1
WiiEvent	ISTUFF
ControlEvent	dac-espc-1
WiiEvent	ISTUFF
ControlEvent	dac-espc-1
WiiEvent	ISTUFF
ControlEvent	dac-espc-1
WiiEvent	ISTUFF
ControlEvent	dac-espc-1
WiiEvent	ISTUFF
ControlEvent	dac-espc-1
WiiEvent	ISTUFF
ControlEvent	dac-espc-1
WiiEvent	ISTUFF
ControlEvent	dac-espc-1

Show	Event Type
✓	ControlEvent
✓	Demo.Initial
✓	PPMapping
✓	PPQuery
✓	PPResponse
✓	WiiEvent

Show	Field Name	Field Type
☐	Action	string
☐	ChainEvent	int
☐	ControlCommand	string
☐	Event Logger Timestamp	long
☐	EventHeapJumps	string
☐	EventHeapVersion	int
✓	EventType	string
☐	IsState	int
☐	MappingEventType	string
☐	MappingString	string
☐	Method	string
☐	NumChunks	int
☐	Result	object
☐	SequenceNum	int
☐	SessionID	int
☐	Source	string
☐	SourceApplication	string
✓	SourceDevice	string
☐	Target	string
☐	TargetApplication	string
☐	TargetDevice	string
☐	TargetGroup	string
☐	TargetPerson	string
☐	TimeToLive	int
☐	WiiCommand	string

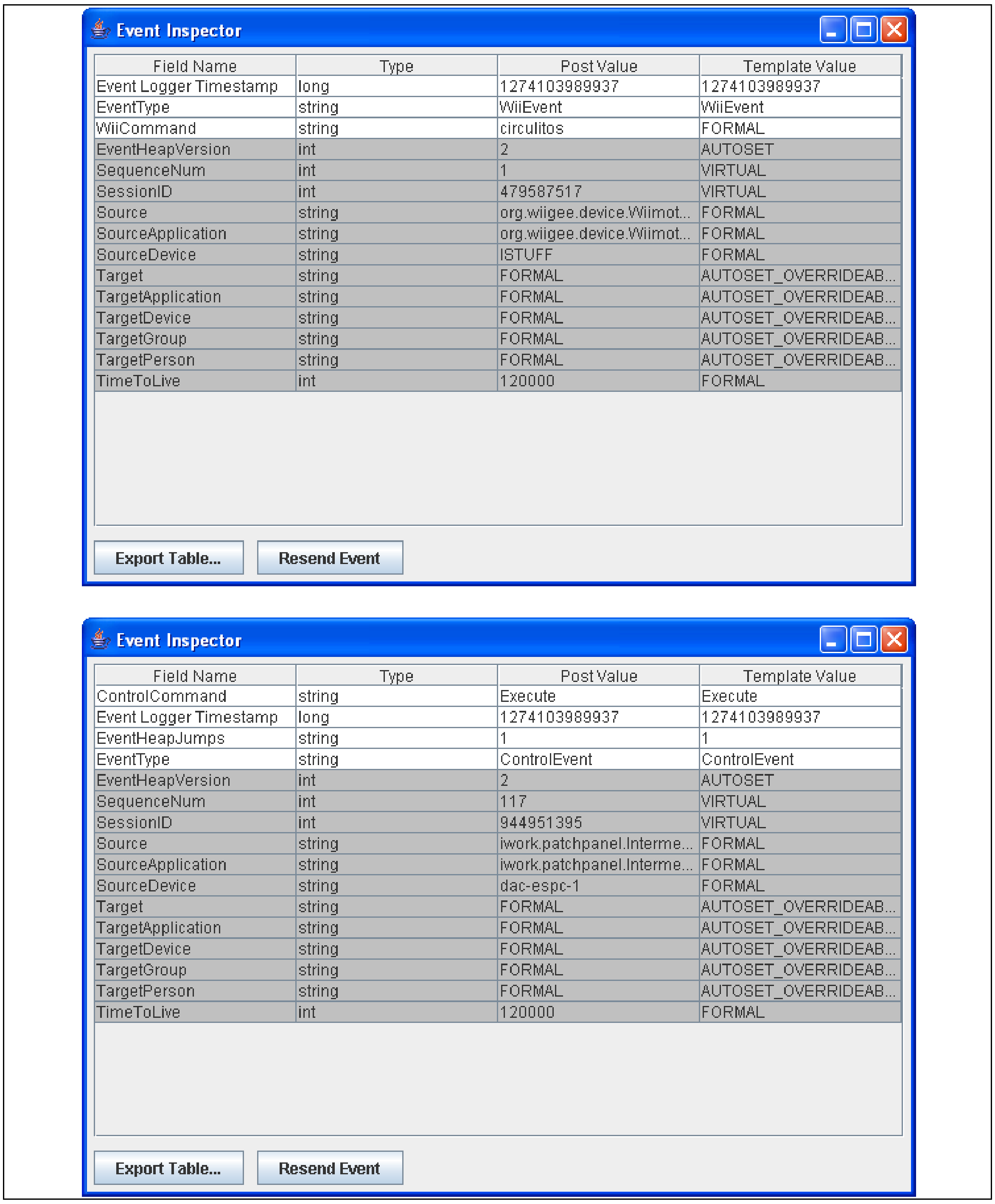
Connected to Event Heap server version 2 on istuff:4535

Fig. 23 Event Logger capture during Controlled Windows Test

Previous Fig. 23 shows the traffic of events into the event heap: we can differentiate between the event sent by wiiGeeGUI (from SourceDevice: ISTUFF) and the corresponding event generated by the patch panel rules (from SourceDevice: dac-espc-1).

The following Fig. 24 shows in more detail two of the packets involved in this process: the first packet is generated by the wiiGee GUI and the second one is the generated by the patch panel rules. The first packet with the “WiiCommand” with post value “circulitos” generates a “ControlCommand” with post value

"Execute". Control Command "Execute" on the Controlled Windows software generates that Java robot simulates the push of "Intro" button.



Field Name	Type	Post Value	Template Value
Event Logger Timestamp	long	1274103989937	1274103989937
EventType	string	WiiEvent	WiiEvent
WiiCommand	string	circulitos	FORMAL
EventHeapVersion	int	2	AUTOSET
SequenceNum	int	1	VIRTUAL
SessionID	int	479587517	VIRTUAL
Source	string	org.wiigee.device.Wiimot...	FORMAL
SourceApplication	string	org.wiigee.device.Wiimot...	FORMAL
SourceDevice	string	ISTUFF	FORMAL
Target	string	FORMAL	AUTOSET_OVERRIDEAB...
TargetApplication	string	FORMAL	AUTOSET_OVERRIDEAB...
TargetDevice	string	FORMAL	AUTOSET_OVERRIDEAB...
TargetGroup	string	FORMAL	AUTOSET_OVERRIDEAB...
TargetPerson	string	FORMAL	AUTOSET_OVERRIDEAB...
TimeToLive	int	120000	FORMAL

Field Name	Type	Post Value	Template Value
ControlCommand	string	Execute	Execute
Event Logger Timestamp	long	1274103989937	1274103989937
EventHeapJumps	string	1	1
EventType	string	ControlEvent	ControlEvent
EventHeapVersion	int	2	AUTOSET
SequenceNum	int	117	VIRTUAL
SessionID	int	944951395	VIRTUAL
Source	string	iwork.patchpanel.Interme...	FORMAL
SourceApplication	string	iwork.patchpanel.Interme...	FORMAL
SourceDevice	string	dac-espc-1	FORMAL
Target	string	FORMAL	AUTOSET_OVERRIDEAB...
TargetApplication	string	FORMAL	AUTOSET_OVERRIDEAB...
TargetDevice	string	FORMAL	AUTOSET_OVERRIDEAB...
TargetGroup	string	FORMAL	AUTOSET_OVERRIDEAB...
TargetPerson	string	FORMAL	AUTOSET_OVERRIDEAB...
TimeToLive	int	120000	FORMAL

Fig. 24 Event sent by WiiGeeGUI and response by Patch Panel

3.3. Collaborative application test results

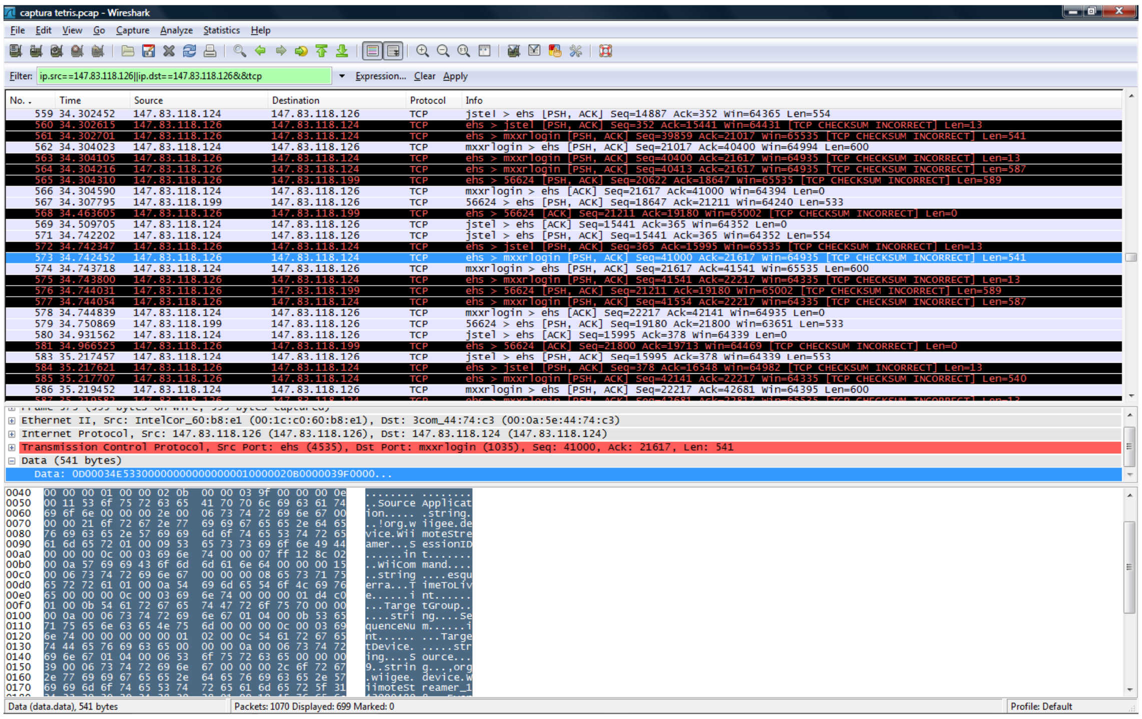
This scenario simulates a real environment in which ubiquitous computing component, iStuff event heap, is connecting to a new human centered device, wiiMote using WiiGee libraries, to interact with a collaborative application: the collaborative Tetris software.

To configure the scenario fist we have to start all istuff components: event heap and event logger applications at host “istuff” and, patch panel and patch panel manager at host “espc-1”. Then we need to load patch panel rules defined in section 3.4.3 using patch panel manager software. The last step consists in startup wiigee software and to connect WiiRemote devices at hosts “jose-pc” and “espc-1” respectively.

The scenario works as following:

1. A user performs a gesture with the WiiRemote device.
2. WiiGee software recognizes (as describes on section 1.6) or not the gesture.
3. IStuff proxy integrated on WiiGee software sends a packet type “WiiEvent” containing the name of the recognized gesture to the event heap.
4. Event heap delivers the packet to patch panel host which captured it.
5. Patch panel looks up in its predefined rules if exists any rule that affects the received packet. In this case, the packet is mapped to another type of packet “ControlEvent” following the rules that you can find in section 3.4.3. The new packet “ControlEvent” with the corresponding value according to the patch panel rules is send again to the event heap. Notice that according to the patch panel rules a “WiiEvent” packet with the same message will generate a different output packet according to the source device that generates the “WiiEvent” packet.
6. Throw the event heap the packet finally arrives to “jose-pc” that contains iStuff proxy (in collaborative Tetris software) that captures this kind of packets.
7. The corresponding event is generated internally in collaborative Tetris software, event listener delivers it to the game and the right Tetris figure move along the square board.

The following Fig. 25 proves the traffic between all the hosts (captured on host “istuff”) is done using TCP connections. All packets are always sent to all the hosts subscribed to event heap.



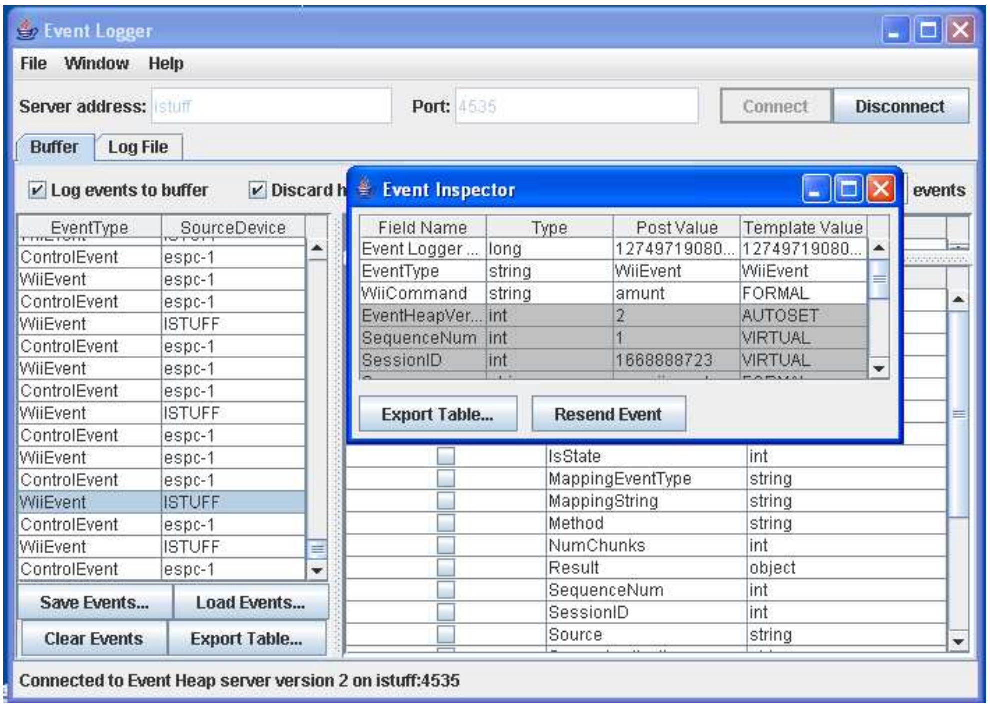
captura tetris.pcap - Wireshark

Filter: ip.src==147.83.118.126||ip.dst==147.83.118.126&&tcp

```
No. .  Time       Source          Destination     Protocol  Info
  559 34.302452  147.83.118.124  147.83.118.126  TCP       jstel > ehs [PSH, ACK] Seq=14887 Ack=352 Win=64365 Len=554
  560 34.302615  147.83.118.126  147.83.118.124  TCP       ehs > jstel [PSH, ACK] Seq=352 Ack=15441 Win=64431 [TCP CHECKSUM INCORRECT] Len=13
  561 34.302701  147.83.118.126  147.83.118.124  TCP       ehs > mxxrlogin [PSH, ACK] Seq=39859 Ack=21017 Win=65535 [TCP CHECKSUM INCORRECT] Len=541
  562 34.304023  147.83.118.124  147.83.118.126  TCP       mxxrlogin > ehs [PSH, ACK] Seq=21017 Ack=40400 Win=64994 Len=600
  563 34.304105  147.83.118.126  147.83.118.124  TCP       ehs > mxxrlogin [PSH, ACK] Seq=40400 Ack=21617 Win=64935 [TCP CHECKSUM INCORRECT] Len=13
  564 34.304216  147.83.118.126  147.83.118.124  TCP       ehs > mxxrlogin [PSH, ACK] Seq=40413 Ack=21617 Win=64935 [TCP CHECKSUM INCORRECT] Len=587
  565 34.304310  147.83.118.126  147.83.118.199  TCP       ehs > 56624 [PSH, ACK] Seq=20622 Ack=18647 Win=65535 [TCP CHECKSUM INCORRECT] Len=589
  566 34.304590  147.83.118.124  147.83.118.126  TCP       mxxrlogin > ehs [ACK] Seq=21617 Ack=41000 Win=64394 Len=0
  567 34.307795  147.83.118.199  147.83.118.126  TCP       56624 > ehs [PSH, ACK] Seq=18647 Ack=21211 Win=64240 Len=533
  568 34.463605  147.83.118.126  147.83.118.199  TCP       ehs > 56624 [ACK] Seq=21211 Ack=19180 Win=65002 [TCP CHECKSUM INCORRECT] Len=0
  569 34.509705  147.83.118.124  147.83.118.126  TCP       jstel > ehs [ACK] Seq=15441 Ack=365 Win=64352 Len=0
  571 34.742202  147.83.118.124  147.83.118.126  TCP       jstel > ehs [PSH, ACK] Seq=15441 Ack=365 Win=64352 Len=554
  572 34.742347  147.83.118.126  147.83.118.124  TCP       ehs > jstel [PSH, ACK] Seq=365 Ack=15995 Win=65535 [TCP CHECKSUM INCORRECT] Len=13
  573 34.742452  147.83.118.126  147.83.118.124  TCP       ehs > mxxrlogin [PSH, ACK] Seq=41000 Ack=21617 Win=64935 [TCP CHECKSUM INCORRECT] Len=541
  574 34.743718  147.83.118.124  147.83.118.126  TCP       mxxrlogin > ehs [PSH, ACK] Seq=21617 Ack=41541 Win=65535 Len=600
  575 34.743800  147.83.118.126  147.83.118.124  TCP       ehs > mxxrlogin [PSH, ACK] Seq=41541 Ack=22217 Win=64335 [TCP CHECKSUM INCORRECT] Len=13
  576 34.744031  147.83.118.126  147.83.118.199  TCP       ehs > 56624 [PSH, ACK] Seq=21211 Ack=19180 Win=65002 [TCP CHECKSUM INCORRECT] Len=589
  577 34.744054  147.83.118.126  147.83.118.124  TCP       ehs > mxxrlogin [PSH, ACK] Seq=41554 Ack=22217 Win=64335 [TCP CHECKSUM INCORRECT] Len=587
  578 34.744839  147.83.118.124  147.83.118.126  TCP       mxxrlogin > ehs [ACK] Seq=22217 Ack=42141 Win=64935 Len=0
  579 34.750869  147.83.118.199  147.83.118.126  TCP       56624 > ehs [PSH, ACK] Seq=19180 Ack=21800 Win=63651 Len=533
  580 34.931562  147.83.118.124  147.83.118.126  TCP       jstel > ehs [ACK] Seq=15995 Ack=378 Win=64339 Len=0
  581 34.966525  147.83.118.126  147.83.118.199  TCP       ehs > 56624 [ACK] Seq=21800 Ack=19713 Win=64469 [TCP CHECKSUM INCORRECT] Len=0
  583 35.217457  147.83.118.124  147.83.118.126  TCP       jstel > ehs [PSH, ACK] Seq=15995 Ack=378 Win=64339 Len=553
  584 35.217621  147.83.118.126  147.83.118.124  TCP       ehs > jstel [PSH, ACK] Seq=378 Ack=16548 Win=64982 [TCP CHECKSUM INCORRECT] Len=13
  585 35.217707  147.83.118.126  147.83.118.124  TCP       ehs > mxxrlogin [PSH, ACK] Seq=42141 Ack=22217 Win=64335 [TCP CHECKSUM INCORRECT] Len=540
  586 35.219452  147.83.118.124  147.83.118.126  TCP       mxxrlogin > ehs [PSH, ACK] Seq=22217 Ack=42681 Win=64395 Len=600

⊞ Ethernet II, Src: IntelCor_60:b8:e1 (00:1c:c0:60:b8:e1), Dst: 3com_44:74:c3 (00:0a:5e:44:74:c3)
⊞ Internet Protocol, Src: 147.83.118.126 (147.83.118.126), Dst: 147.83.118.124 (147.83.118.124)
⊞ Transmission Control Protocol, Src Port: ehs (4535), Dst Port: mxxrlogin (1035), Seq: 41000, Ack: 21617, Len: 541
⊟ Data (541 bytes)
    Data: 0D00034E5330000000000000000010000020B0000039F0000...

0040  00 00 00 01 00 00 02 0b  00 00 03 9f 00 00 00 0e   ........ ........
0050  00 11 53 6f 75 72 63 65  41 70 70 6c 69 63 61 74   ..Source Applicat
0060  69 6f 6e 00 00 00 2e 00  06 73 74 72 69 6e 67 00   ion..... .string.
0070  00 00 21 6f 72 67 2e 77  69 69 67 65 65 2e 64 65   ..!org.w iigee.de
0080  76 69 63 65 2e 57 69 69  6d 6f 74 65 53 74 72 65   vice.Wii moteStre
0090  61 6d 65 72 01 00 09 53  65 73 73 69 6f 6e 49 44   amer...S essionID
00a0  00 00 00 0c 00 03 69 6e  74 00 00 07 ff 12 8c 02   ......in t.......
00b0  00 0a 57 69 69 43 6f 6d  6d 61 6e 64 00 00 00 15   ..WiiCom mand....
00c0  00 06 73 74 72 69 6e 67  00 00 00 08 65 73 71 75   ..string ....esqu
00d0  65 72 72 61 01 00 0a 54  69 6d 65 54 6f 4c 69 76   erra...T imeToLiv
00e0  65 00 00 00 0c 00 03 69  6e 74 00 00 00 01 d4 c0   e......i nt......
00f0  01 00 0b 54 61 72 67 65  74 47 72 6f 75 70 00 00   ...Targe tGroup..
0100  00 0a 00 06 73 74 72 69  6e 67 01 04 00 0b 53 65   ....stri ng....Se
0110  71 75 65 6e 63 65 4e 75  6d 00 00 00 0c 00 03 69   quenceNu m......i
0120  6e 74 00 00 00 00 00 01  02 00 0c 54 61 72 67 65   nt...... ...Targe
0130  74 44 65 76 69 63 65 00  00 00 0a 00 06 73 74 72   tDevice. .....str
0140  69 6e 67 01 04 00 06 53  6f 75 72 63 65 00 00 00   ing....S ource...
0150  39 00 06 73 74 72 69 6e  67 00 00 00 2c 6f 72 67   9..strin g...,org
0160  2e 77 69 69 67 65 65 2e  64 65 76 69 63 65 2e 57   .wiigee. device.W
0170  69 69 6d 6f 74 65 53 74  72 65 61 6d 65 72 5f 31   iimoteSt reamer_1

Data (data.data), 541 bytes    Packets: 1070 Displayed: 699 Marked: 0    Profile: Default
```

Fig. 25 Traffic between the hosts involved on the third test

We are going to observe in more understandable point of view the traffic that event heap supports using the event logger and event inspector applications; tools that are present in iStuff toolkit.



Event Logger

Server address: istuff   Port: 4535   Connect   Disconnect

Buffer | Log File

☑ Log events to buffer   ☑ Discard h... events

EventType	SourceDevice
ControlEvent	espc-1
WiiEvent	espc-1
ControlEvent	espc-1
WiiEvent	ISTUFF
ControlEvent	espc-1
WiiEvent	espc-1
ControlEvent	espc-1
WiiEvent	ISTUFF
ControlEvent	espc-1
WiiEvent	espc-1
ControlEvent	espc-1
WiiEvent	ISTUFF
ControlEvent	espc-1
WiiEvent	ISTUFF
ControlEvent	espc-1

Save Events... | Load Events... | Clear Events | Export Table...

Event Inspector

Field Name	Type	Post Value	Template Value
Event Logger ...	long	12749719080...	12749719080...
EventType	string	WiiEvent	WiiEvent
WiiCommand	string	amunt	FORMAL
EventHeapVer...	int	2	AUTOSET
SequenceNum	int	1	VIRTUAL
SessionID	int	1668888723	VIRTUAL

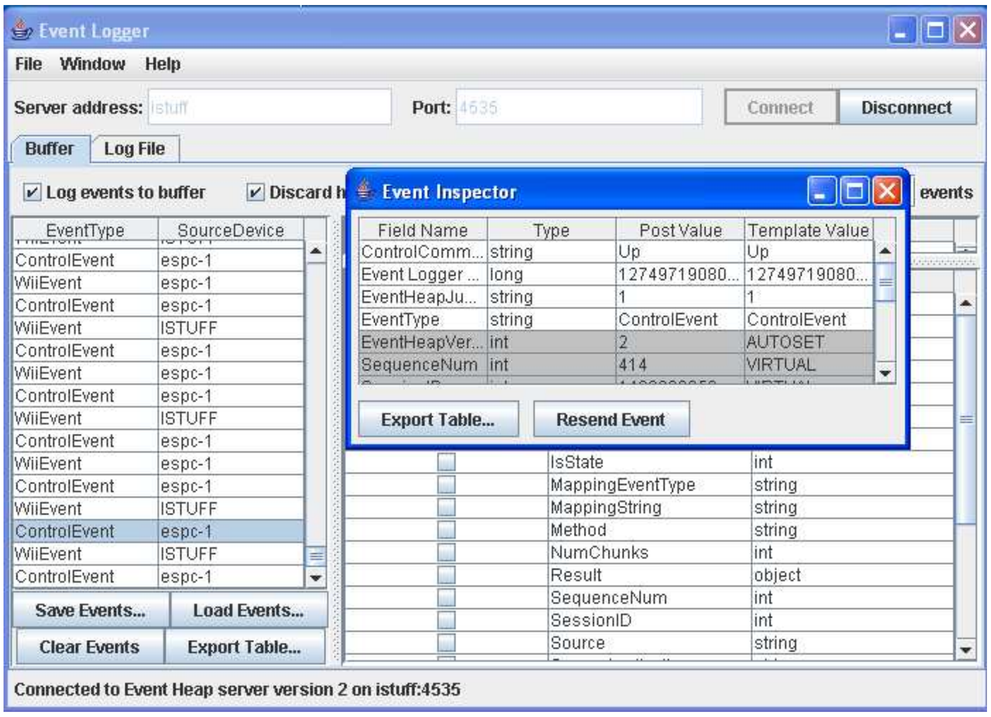
Export Table... | Resend Event

☐	IsState	int
☐	MappingEventType	string
☐	MappingString	string
☐	Method	string
☐	NumChunks	int
☐	Result	object
☐	SequenceNum	int
☐	SessionID	int
☐	Source	string

Connected to Event Heap server version 2 on istuff:4535

Fig. 26 WiiEvent packet generated by WiiGee software on istuff host

The previous Fig. 26 shows a packet send by the iStuff proxy integrated on WiiGee software that runs on istuff host. This packet is generated when a gesture type “amunt” (present on the Wii Comand field name on the “Post Value” row in Event Inspector window) is recognized by WiiGee software.



EventType	SourceDevice
ControlEvent	espc-1
WiiEvent	espc-1
ControlEvent	espc-1
WiiEvent	ISTUFF
ControlEvent	espc-1
WiiEvent	espc-1
ControlEvent	espc-1
WiiEvent	ISTUFF
ControlEvent	espc-1
WiiEvent	espc-1
ControlEvent	espc-1
WiiEvent	ISTUFF
ControlEvent	espc-1
WiiEvent	ISTUFF
ControlEvent	espc-1

Field Name	Type	Post Value	Template Value
ControlComm...	string	Up	Up
Event Logger ...	long	12749719080...	12749719080...
EventHeapJu...	string	1	1
EventType	string	ControlEvent	ControlEvent
EventHeapVer...	int	2	AUTOSET
SequenceNum	int	414	VIRTUAL

☐	IsState	int
☐	MappingEventType	string
☐	MappingString	string
☐	Method	string
☐	NumChunks	int
☐	Result	object
☐	SequenceNum	int
☐	SessionID	int
☐	Source	string

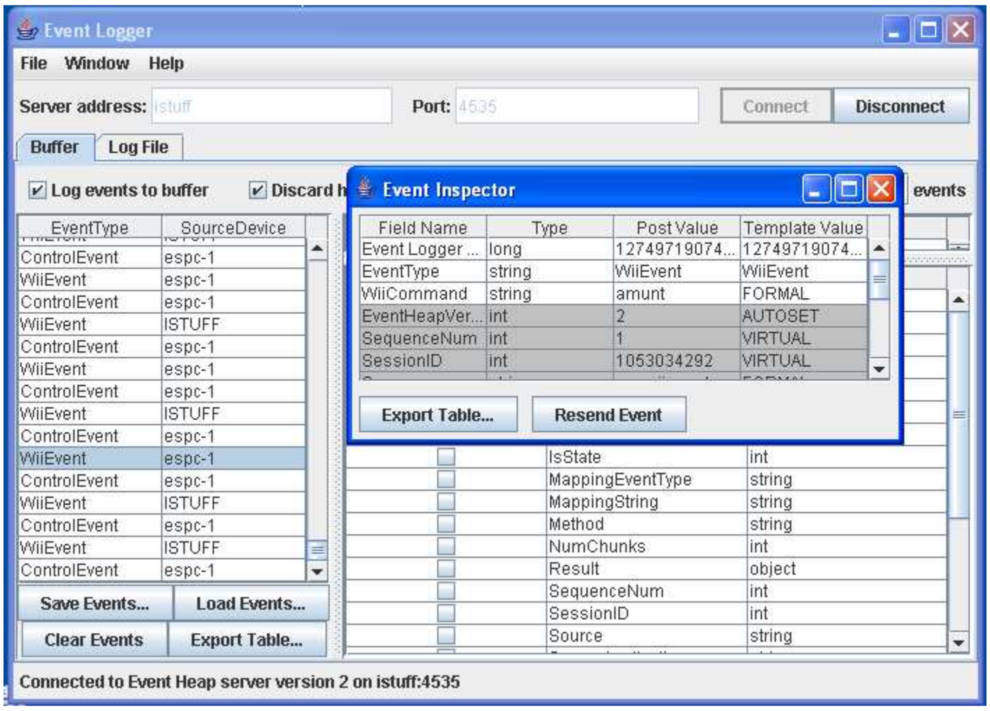
Connected to Event Heap server version 2 on istuff:4535

Fig. 27 ControlEvent packet (Up) generated by patch panel software on espc-1 host

Previous Fig. 27 shows the packet generated by patch panel on host “espc-1” as a response on the previous packet shown on figure 26. Event Inspector details the content of the packet where we can observe that the previous packet type “WiiComand” with post value “amunt” generated a “ControlCommand” packet with “Up” value. This type of packet produces that player one figure move all down in the square board.

The following Fig. 28 shows a same type of packet but now, generated by espc-1 host with which plays player 2.

We can observe that packet contains the same Event type value because the version software used is the same as in “istuff” host.



Event Logger

File Window Help

Server address: istuff Port: 4535 Connect Disconnect

Buffer Log File

☑ Log events to buffer ☑ Discard h

EventType	SourceDevice
ControlEvent	espc-1
WiiEvent	espc-1
ControlEvent	espc-1
WiiEvent	ISTUFF
ControlEvent	espc-1
WiiEvent	espc-1
ControlEvent	espc-1
WiiEvent	ISTUFF
ControlEvent	espc-1
WiiEvent	espc-1
ControlEvent	espc-1
WiiEvent	ISTUFF
ControlEvent	espc-1
WiiEvent	ISTUFF
ControlEvent	espc-1

Save Events... Load Events... Clear Events Export Table...

Event Inspector

Field Name	Type	Post Value	Template Value
Event Logger ...	long	12749719074...	12749719074...
EventType	string	WiiEvent	WiiEvent
WiiCommand	string	amunt	FORMAL
EventHeapVer...	int	2	AUTOSET
SequenceNum	int	1	VIRTUAL
SessionID	int	1053034292	VIRTUAL

Export Table... Resend Event

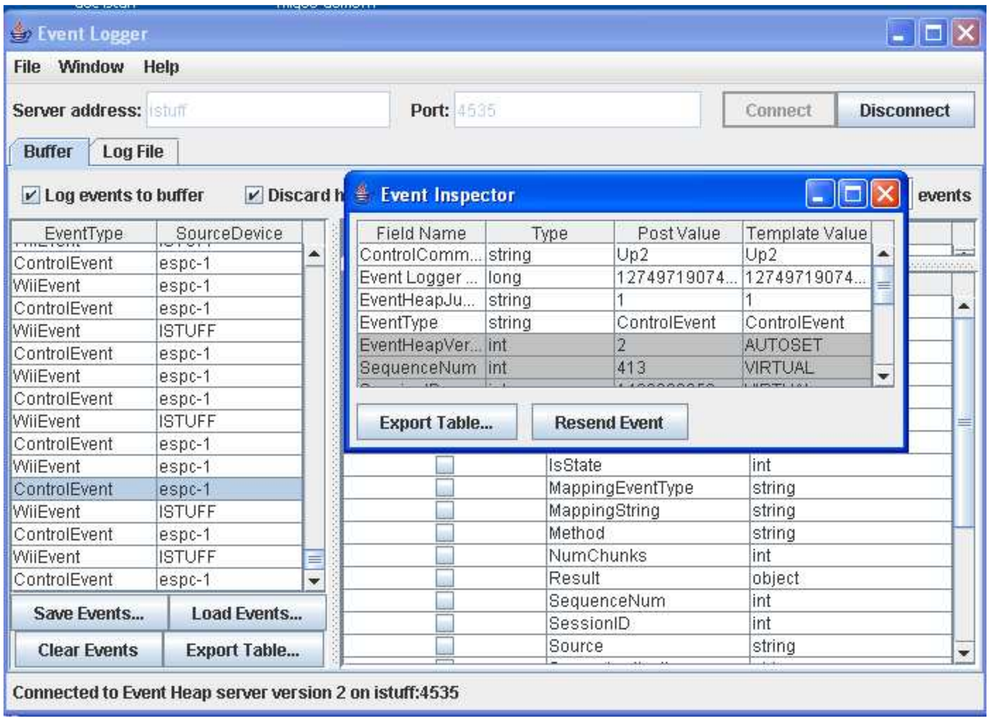
events

☐	IsState	int
☐	MappingEventType	string
☐	MappingString	string
☐	Method	string
☐	NumChunks	int
☐	Result	object
☐	SequenceNum	int
☐	SessionID	int
☐	Source	string

Connected to Event Heap server version 2 on istuff:4535

Fig. 28 WiiEvent packet generated by WiiGee software on espc-1 host

Following Fig. 29 shows a Control Event packet with "Up2" PostValue which in this case is also generated by patch panel software that runs on espc-1 host. Notice that in this case post value changed compared with the packet shown in Fig. 27. This is due to the rules loaded in patch panel; it has detected that in this case the packet comes from "espc-1" host and not from "istuff" host.



Event Logger

File Window Help

Server address: istuff Port: 4535 Connect Disconnect

Buffer Log File

☑ Log events to buffer ☑ Discard h

EventType	SourceDevice
ControlEvent	espc-1
WiiEvent	espc-1
ControlEvent	espc-1
WiiEvent	ISTUFF
ControlEvent	espc-1
WiiEvent	espc-1
ControlEvent	espc-1
WiiEvent	ISTUFF
ControlEvent	espc-1
WiiEvent	espc-1
ControlEvent	espc-1
WiiEvent	ISTUFF
ControlEvent	espc-1
WiiEvent	ISTUFF
ControlEvent	espc-1

Save Events... Load Events... Clear Events Export Table...

Event Inspector

Field Name	Type	Post Value	Template Value
ControlComm...	string	Up2	Up2
Event Logger ...	long	12749719074...	12749719074...
EventHeapJu...	string	1	1
EventType	string	ControlEvent	ControlEvent
EventHeapVer...	int	2	AUTOSET
SequenceNum	int	413	VIRTUAL

Export Table... Resend Event

events

☐	IsState	int
☐	MappingEventType	string
☐	MappingString	string
☐	Method	string
☐	NumChunks	int
☐	Result	object
☐	SequenceNum	int
☐	SessionID	int
☐	Source	string

Connected to Event Heap server version 2 on istuff:4535

Fig. 29 ControlEvent packet (Up2) generated by patch panel software on espc-1 host

CHAPTER 4. Conclusions

This chapter contains the master thesis conclusions. First the test conclusions are present. Following the green study about the possible impact of studied technology could produce in the environment, and the personal conclusions I have obtained carrying out this master thesis.

4.1 Tests conclusions

iStuff toolkit was created to act as a middleware allows interconnection between a high number of devices that initially, where not created to interact in an ubiquitous environment. In our case, has permitted successfully the interconnection between Nintendo Wii Remote devices and software on different hosts with different operating system.

The first test has shown that iStuff software works successfully according to the programmer specifications and the basic scenario configuration is correct.

The second test has proved that iStuff proxy created to interact with Event Heap works successfully and also proved a simple way to interact with an operating system using a WiiMote device.

The last test represented how to introduce some “intelligence” to the patch panel’s iStuff toolkit from the differentiation between the source host that generates a specific packet and another host that generates the same packet type. Also has shown how to integrate ubiquitous components into an existing program to become part of “ubiquitous environment”.

We can achieve the following points from iStuff toolkit experience:

- Fast and easy: iStuff toolkit architecture has proved as a fast and easy environment to develop applications that can interact in a ubiquitous environment. Only with the creation of a proxy that acts as middleware between our applications or our hardware devices firmware we can achieve a ubiquitous device.
- Interoperability: Thanks to the use of Java technology iStuff toolkit is independent of the operating system. Besides, it would be used in tiny or small operating systems (such as firmwares) that implement a Java Virtual Machine on its systems.
- Independent: Using hostname instead of IP addresses permits the connection between hosts across Internet and independent from static or dynamic IP addresses. It stands up the point that in ubiquitous environment network elements are dynamic and can appear or disappear in the network, fact that using DNS can be solved correctly.

- Multiuser: Different users can interact at same time with iStuff toolkit independently or together, in a different or in the same task without any appreciable difference.
- Security: IStuff toolkit doesn’t give any kind of security to the users. All the TCP packets between the hosts travel “in clear” through the network. Neither any security check is done when patch panel rules are set up. Both security fails could permit that a malicious user manages the network as he pleased.
- Limitations: Java Virtual Machine has become unstable in some moments due to the suddenly shut down of some iStuff components. In those cases a reboot of virtual machine was needed.

Another device and related software has been used in this master thesis: the Nintendo’s Wii Remote. It was plugged into personal computer environment through WiiGee java libraries. Wii Remote has demonstrated a powerful device in which new human-to-computer interfaces refers. It could be a starting point, for example, to bring closer computer technology to elderly population or to people with a reduced accessibility.

The main limitation we noticed is the gesture recognition procedure: a gesture that could be easy to reproduce for a person couldn’t be for another person

This is, when a user tries to use WiiRemote with the predefined gestures of another user, some gesture will be difficult to reproduce. The solution goes through the individual training of gestures for each user involved. This fact doesn’t’ represent a hint from the point of view of the software because WiiGee GUI permits save and load gestures set during execution time.

To sum up, this master thesis has accomplished the proposed goals. It has proved the capabilities of wiiGee library with some limitations when two gestures are very similar, it can confuse wiiGee software and recognize a wrong gesture. Especially you can notice it when saved gestures from one user are tried to be reproduced by another user.

Also iStuff toolkit has proved as a fast and easy environment to create ubiquitous test environment. Patch Panel element has demonstrated as a powerful tool because allows creation of complex mapping between events. An iStuff proxy has been integrated successfully into wiiGeeGUI allowing us to create a ubiquitous environment. To complete this environment a collaborative application has been programmed. That application permits the interaction between two users and the ubiquitous scenario elements with a common objective: complete a Tetris game.

4.2 Green study

The execution of this master thesis did not represent any harmful effect to the environment excluding the possible responsibilities during the manufacturing process of the hardware devices used; so this kind of responsibilities are not of our scope.

4.3. Personal conclusions

When I started this master thesis I realized that ubiquitous computing is one of the next steps in human to computer interaction refers. This is, don’t learn how to use computer technology, transform it to technology that learns from users how they want devices act.

Also exists a related topic not discussed in this master thesis but not less important. That is the associated anonymity of the users of a ubiquitous environment. Imagine that undesired user access to your personal ubiquitous home network; he would know what TV program you are watching, access perhaps to your personal photos or simply locate yourself inside the building. I think to try to avoid this kind of problems, future ubiquitous environments will emphasize security and anonimity related issues.

Personally, I will try to develop my personal career in one of the subjects I studied during the execution of this mater thesis, ubiquitous computing and/or new human to computer interfaces. Those subjects are still not present in our daily live and could be an emerging market. Another interesting point is those technologies apologizes to facilitate users lives and society‘s well done.

GLOSSARY

DNS – Domain Name Server

GUI – Graphical User Interface

HCI – Human - Computer Interaction

ID – Identification

L2CAP – Logical Link Control and Adaptation Protocol

OED – Oxford English Dictionary

OS – Operating System

PDA – Personal Digital Assistant

SDK – Software Development Kit

TCP – Transmission Control Protocol

VNC – Virtual Network Computing

REFERENCES

[1] Oxford University Press. Oxford English Dictionary. Oxford University Press, 2000.

[2] *Stajano, Frank*. Security for ubiquitous computing, John Willey & Sons, Ltd. 2002

[3] http://www.cl.cam.ac.uk/research/dtg/attarchive/ab.html

[4] http://www.cl.cam.ac.uk/research/dtg/research/wiki/BatSystem

[5] http://oxygen.csail.mit.edu/Overview.html

[6] http://en.wikipedia.org/wiki/User-centered_design

[7] http://en.wikipedia.org/wiki/Human-centered_computing_(discipline)

[8] http://hci.stanford.edu/research/istuff.html

[9] http://en.wikipedia.org/wiki/Human%E2%80%93computer_interaction

[10] *Rafael Ballagas, Meredith Ringel, Maureen Stone, Jan Borchers*. iStuff: A Physical User Interface Toolkit for Ubiquitous Computing Environments

[11] *Jan Borchers, Meredith Ringel, Joshua Tyler, and Armando Fox*. Stanford Interactive Workspaces: A Framework for Physical and Graphical User Interface Prototyping - http://www.cl.cam.ac.uk/research/dtg/research/wiki/BatSystem

[12] http://en.wikipedia.org/wiki/X10_%28industry_standard%29

[13] http://wiibrew.org/wiki/Wiimote

[14] http://www.wiigee.org/

[15] http://java3d.dev.java.net/

[16] http://us.kensington.com/html/9403.html

[17] http://www.taringa.net/posts/downloads/1533488/Como-instalar-WIDCOMM-drivers-5_1_0_1100.html

[18] http://markmail.org/message/izwhoja2pma6a3vx#query:wiiremote%20widcom%20l2cap+page:1+mid:4y5w44e5enbmc7ok+state:results

[19] http://www.cs.unc.edu/~dewan/290/s97/notes/intro/node2.html

[20] http://www.broadcom.com/products/Bluetooth/Bluetooth-RF-Silicon-and-Software-Solutions/BCM2042

[21] http://www.analog.com/static/imported-files/data_sheets/ADXL330.pdf

[22] http://www.wiigee.org/download_files/gesture_recognition_with_a_wii_controller-schloemer_poppinga_henze_boll.pdf

Annex 1

TerminalEventSender.java

```
import iwork.eheap2.*;
import java.net.*;
import java.util.*;
import java.io.Serializable;

public class TerminalEventSender{
      private EventHeap mEventHeap;
      public String mMachineName;

      public TerminalEventSender(){
            try{
                  String eventHeapName = "istuff";
                  String eventType = "hello";
                  List<Object> fieldList = new ArrayList<Object>();
                  /**for (int i=2;i<args.length; i++){
                        fieldList.add(new FieldValuePair(args[i]) );

                  }**/
                  System.out.println("sending...");
                  send(eventHeapName, eventType, fieldList);
            }
            catch (ArrayIndexOutOfBoundsException aexp){
                  printUsage();
            }
      }

      public void send(String iEventHeapName, String iEventType, List
iFieldValueList){
            mEventHeap = new EventHeap(iEventHeapName);
            try{
                  Event oEvent = new Event(iEventType);
                  for (Iterator i=iFieldValueList.iterator();
i.hasNext();){
                        FieldValuePair pair = (FieldValuePair)i.next();
                        oEvent.setField(pair.mFieldName,  pair.mValue,
FieldValueTypes.FORMAL);
                  }
                  System.out.println("putting event...");
                  mEventHeap.putEvent(oEvent);
            }
            catch (EventHeapException eexp){
            }
      }

      //Integer Number > Float Number > String
      class FieldValuePair{
            String mFieldName;
            Serializable mValue;
            public FieldValuePair(String iStringToParse){
                  //ffasd:asdda
                  //ssasdsdf:
                  //"ssdasda asdasd":
                  String[] parts;
                  if (iStringToParse.startsWith("\"")){
                        parts = iStringToParse.split("\\\":");
                  }
                  else{
                        parts = iStringToParse.split(":");
                  }

                  String iFieldName = parts[0].trim();
                  if (iFieldName.startsWith("\"")){  //Strip off quotes
                        iFieldName = 
iFieldName.substring(1,iFieldName.length());
                  }

                  String iValString = "";
                  if (parts.length>0){
                        iValString= parts[1].trim();
                  }

                  mFieldName = iFieldName;
                  try{
                        int intValue = 
Integer.parseInt(iValString.trim());
                        mValue = new Integer(intValue);
                  }
                  catch (NumberFormatException niExp){
                        try{
                              float floatValue = 
Float.parseFloat(iValString.trim());
                              mValue = new Float(floatValue);
                        }
                        catch (NumberFormatException nfExp){
                              mValue = iValString;
                        }//enf try float
                  }//end try integer
            }
      }


     public static void main(String[] args)
     {
                  TerminalEventSender s;
                  s = new TerminalEventSender();

     }


      public static void printUsage(){
            System.out.println("Usage: TerminalEventSender <event heap 
name> <event type> {<field name:value>}*");
            System.exit(-1);
      }


}
```

Annex 2

IStuffProxy.java

```
package wiigeegui;

import iwork.eheap2.Event;
import iwork.eheap2.EventHeap;
import iwork.eheap2.EventHeapException;
import iwork.eheap2.FieldValueTypes;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

public class IStuffProxy {
    private EventHeap mEventHeap;
    public String mMachineName;
    String eventHeapName = "localhost";
    String eventType = "WiiEvent";
    List<Object> fieldList = new ArrayList<Object>();

    public void arrenca(String command) {
        fieldList.add(new FieldValuePair("WiiCommand:"+command));

        System.out.println("sending...");
        send(eventHeapName, eventType, fieldList);
    }

    public void send(String iEventHeapName, String iEventType,
            List iFieldValueList) {
        mEventHeap = new EventHeap(iEventHeapName);
        try {
            Event oEvent = new Event(iEventType);
            for (Iterator i = iFieldValueList.iterator();
i.hasNext();) {
                FieldValuePair pair = (FieldValuePair) i.next();
                oEvent.setField(pair.mFieldName, pair.mValue,
                        FieldValueTypes.FORMAL);
            }
            System.out.println("putting event...");
            mEventHeap.putEvent(oEvent);
        } catch (EventHeapException eexp) {
        }
    }

    // Integer Number > Float Number > String
    class FieldValuePair {
        String mFieldName;
        Serializable mValue;

        public FieldValuePair(String iStringToParse) {
            // ffasd:asdda
            // ssasdsdf:
```

```
            // "ssdasda asdasd":
            String[] parts;
            if (iStringToParse.startsWith("\"")) {
                parts = iStringToParse.split("\\\":");
            } else {
                parts = iStringToParse.split(":");
            }

            String iFieldName = parts[0].trim();
            if (iFieldName.startsWith("\"")) { // Strip off quotes
                iFieldName = iFieldName.substring(1,
iFieldName.length());
            }

            String iValString = "";
            if (parts.length > 0) {
                iValString = parts[1].trim();
            }

            mFieldName = iFieldName;
            try {
                int intValue = Integer.parseInt(iValString.trim());
                mValue = new Integer(intValue);
            } catch (NumberFormatException niExp) {
                try {
                    float floatValue =
Float.parseFloat(iValString.trim());
                    mValue = new Float(floatValue);
                } catch (NumberFormatException nfExp) {
                    mValue = iValString;
                }// enf try float
            }// end try integer
        }
    }
}
```

ControlledWindows.java

```
package src;


import java.awt.AWTException;
import java.awt.Robot;
import java.awt.event.KeyEvent;

import iwork.eheap2.EventHeap;



public class ControledWindows implements Runnable {
    iwork.eheap2.Event templateEvent;
    EventHeap m_EventHeap;

    ControledWindows(String server)
    {
        super();
System.out.println ("Trying to connect\n");
        m_EventHeap = new EventHeap(server);
      System.out.println ("Did we connect?");
        try {
            templateEvent = new iwork.eheap2.Event("ControlEvent");
        } catch(Exception ex) { ex.printStackTrace(); }
        //setDaemon(true);
    }

    public void run()
    {
        while(true) {
            try {
                iwork.eheap2.Event e =
m_EventHeap.waitForEvent(templateEvent);
                handleEventHeapEvent(e);
            } catch(Exception ex) { ex.printStackTrace(); }
        }
    }

    public void handleEventHeapEvent(iwork.eheap2.Event e)
    {
      Robot rob = null;
      try {
                  rob = new Robot();
            } catch (AWTException e1) {
                  // TODO Auto-generated catch block
                  e1.printStackTrace();
            }

        try {
            String command = e.getPostValueString("ControlCommand");
          System.out.println("command: " + command);
          if (command.equals("Windows")) {
            rob.keyPress(KeyEvent.VK_WINDOWS);
            rob.keyRelease(KeyEvent.VK_WINDOWS);
          } else if (command.equals("Left")) {
            rob.keyPress(KeyEvent.VK_LEFT);
                  rob.keyRelease(KeyEvent.VK_LEFT);
          } else if (command.equals("Right")) {
            rob.keyPress(KeyEvent.VK_RIGHT);
                  rob.keyRelease(KeyEvent.VK_RIGHT);
          } else if (command.equals("Up")) {
            rob.keyPress(KeyEvent.VK_UP);
                  rob.keyRelease(KeyEvent.VK_UP);
          } else if (command.equals("Down")) {
            rob.keyPress(KeyEvent.VK_DOWN);
                  rob.keyRelease(KeyEvent.VK_DOWN);
          } else if (command.equals("Execute")) {
            rob.keyPress(KeyEvent.VK_ENTER);
                  rob.keyRelease(KeyEvent.VK_ENTER);
          }

        } catch(Exception ex) { ex.printStackTrace(); }

    }

    public static void main(String [] argv) {
      ControledWindows cw = new ControledWindows("istuff");
      Thread t = new Thread(cw);
      t.start();
    }
```

Annex 3

Game.java

```java
/*
 * @(#)Game.java
 *
 * This work is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License as
 * published by the Free Software Foundation; either version 2 of
 * the License, or (at your option) any later version.
 *
 * This work is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * Copyright (c) 2003 Per Cederberg. All rights reserved.
 */

package net.tetriscollaborative;

import java.awt.Button;
import java.awt.Component;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Insets;
import java.awt.Label;
import java.awt.Rectangle;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;

/**
 * The Tetris game. This class controls all events in the game and
 * handles all the game logics. The game is started through user
 * interaction with the graphical game component provided by this
 * class.
 *
 * @version  1.2
 * @author   Per Cederberg, per@percederberg.net
 */
public class Game extends Object implements
IstuffProxy.ControlListener {

    /**
     * The main square board. This board is used for the game itself.
     */
    private SquareBoard board = null;

    /**
     * The preview square board. This board is used to display a
     * preview of the figures.
     */
    private SquareBoard previewBoard = new SquareBoard(5, 5);
    /**
     * preview x la peça 2
     */
    private SquareBoard previewBoard2 = new SquareBoard(5, 5);

    /**
     * The figures used on both boards. All figures are reutilized in
     * order to avoid creating new objects while the game is running.
     * Special care has to be taken when the preview figure and the
     * current figure refers to the same object.
     */
    private Figure[] figures = {
        new Figure(Figure.SQUARE_FIGURE),
        new Figure(Figure.LINE_FIGURE),
        new Figure(Figure.S_FIGURE),
        new Figure(Figure.Z_FIGURE),
        new Figure(Figure.RIGHT_ANGLE_FIGURE),
        new Figure(Figure.LEFT_ANGLE_FIGURE),
        new Figure(Figure.TRIANGLE_FIGURE)
    };

    private Figure[] figures2 = {
            new Figure(Figure.SQUARE_FIGURE),
            new Figure(Figure.LINE_FIGURE),
            new Figure(Figure.S_FIGURE),
            new Figure(Figure.Z_FIGURE),
            new Figure(Figure.RIGHT_ANGLE_FIGURE),
            new Figure(Figure.LEFT_ANGLE_FIGURE),
            new Figure(Figure.TRIANGLE_FIGURE)
        };

    /**
     * The graphical game component. This component is created on the
     * first call to getComponent().
     */
    private GamePanel component = null;

    /**
     * The thread that runs the game. When this variable is set to
     * null, the game thread will terminate.
     */
    private GameThread thread = null;

    /**
     * The game level. The level will be increased for every 20 lines
     * removed from the square board.
     */
    private int level = 1;

    /**
     * The current score. The score is increased for every figure that
     * is possible to place on the main board.
     */
    private int score = 0;

    /**
     * The current figure. The figure will be updated when
     */
    private Figure figure = null;
    /**
     * figura 2
     */
    private Figure figure2 = null;

    /**
     * The next figure.
     */
    private Figure nextFigure = null;

    private Figure nextFigure2 = null;

    /**
     * The rotation of the next figure.
     */
    private int nextRotation = 0;

    private int nextRotation2 = 0;

    /**
     * The figure preview flag. If this flag is set, the figure
     * will be shown in the figure preview board.
     */
    private boolean preview = true;

    /**
     * The move lock flag. If this flag is set, the current figure
     * cannot be moved. This flag is set when a figure is moved all
     * the way down, and reset when a new figure is displayed.
     */
    private boolean moveLock = false;

    private boolean moveLock2 = false;
    /**
     * Creates a new Tetris game. The square board will be given
     * the default size of 10x20.
     */
    public Game() {
        this(10, 20);
    }

    /**
     * Creates a new Tetris game. The square board will be given
     * the specified size.
     *
     * @param width     the width of the square board (in positions)
     * @param height    the height of the square board (in positions)
     */
    public Game(int width, int height) {
        board = new SquareBoard(width, height);
        board.setMessage("Press start");
        thread = new GameThread();
    }

    /**
     * Kills the game running thread and makes necessary clean-up.
     * After calling this method, no further methods in this class
     * should be called. Neither should the component returned
     * earlier be trusted upon.
     */
    public void quit() {
        thread = null;
    }

    /**
     * Returns a new component that draws the game.
     *
     * @return the component that draws the game
     */
    public Component getComponent() {
        if (component == null) {
            component = new GamePanel();
        }
        return component;
    }

    /**
     * Handles a game start event. Both the main and preview square
     * boards will be reset, and all other game parameters will be
     * reset. Finally the game thread will be launched.
     */
    private void handleStart() {

        // Reset score and figures
        level = 1;
        score = 0;

        figure = null;
        nextFigure = randomFigure();
        nextFigure.rotateRandom();
        nextRotation = nextFigure.getRotation();

        figure2 = null;
        nextFigure2 = randomFigure2();
        nextFigure2.rotateRandom();
        nextRotation2 = nextFigure2.getRotation();

        // Reset components
        board.setMessage(null);
        board.clear();
        previewBoard.clear();
        handleLevelModification();
        handleScoreModification();
        component.button.setLabel("Pause");

        // Start game thread
        thread.reset();
    }

    /**
     * Handles a game over event. This will stop the game thread,
     * reset all figures and print a game over message.
     */
    private void handleGameOver() {

        // Stop game thred
        thread.setPaused(true);

        // Reset figures
        if (figure != null) {
            figure.detach();
        }
        if (figure2 != null){
            figure2.detach();
        }
        figure = null;
        figure2 = null;

        if (nextFigure != null) {
            nextFigure.detach();
        }
        if (nextFigure2 != null) {
            nextFigure2.detach();
        }
        nextFigure = null;
        nextFigure2 = null;

        // Handle components
        board.setMessage("Game Over");
        component.button.setLabel("Start");
    }

    /**
     * Handles a game pause event. This will pause the game thread and
     * print a pause message on the game board.
     */
    private void handlePause() {
        thread.setPaused(true);
        board.setMessage("Paused");
        component.button.setLabel("Resume");
    }

    /**
     * Handles a game resume event. This will resume the game thread
     * and remove any messages on the game board.
     */
    private void handleResume() {
        board.setMessage(null);
        component.button.setLabel("Pause");
        thread.setPaused(false);
    }

    /**
     * Handles a level modification event. This will modify the level
     * label and adjust the thread speed.
     */
    private void handleLevelModification() {
        component.levelLabel.setText("Level: " + level);
        thread.adjustSpeed();
    }

    /**
     * Handle a score modification event. This will modify the score
     * label.
     */
    private void handleScoreModification() {
        component.scoreLabel.setText("Score: " + score);
    }

    /**
     * Handles a figure start event. This will move the next figure
     * to the current figure position, while also creating a new
     * preview figure. If the figure cannot be introduced onto the
     * game board, a game over event will be launched.
     */
    private void handleFigureStart() {
        int  rotation;

        int rotation2;

        moveLock = false;
        moveLock2 = false;

        // Move next figure to current
        figure = nextFigure;

        rotation = nextRotation;
        nextFigure = randomFigure();
        nextFigure.rotateRandom();
        nextRotation = nextFigure.getRotation();


        figure2 = nextFigure2;
        rotation2 = nextRotation2;
        nextFigure2 = randomFigure2();
        nextFigure2.rotateRandom();
        nextRotation2 = nextFigure2.getRotation();

        // Handle figure preview
        if (preview) {
            previewBoard.clear();
            nextFigure.attach(previewBoard, true,false);
            nextFigure.detach();
            previewBoard2.clear();
            nextFigure2.attach(previewBoard2, true,false);
            nextFigure2.detach();
        }

        // Attach figure to game board
        figure.setRotation(rotation);
        if (!figure.attach(board, false,false)) {
            previewBoard.clear();
            figure.attach(previewBoard, true,false);
            figure.detach();
            handleGameOver();
        }
        figure2.setRotation(rotation2);
        if (!figure2.attach(board, false,true)) {
            previewBoard2.clear();
            figure2.attach(previewBoard2, true,false);
            figure2.detach();
            handleGameOver();
        }

    }

    /**
     * Handles a figure landed event. This will check that the figure
     * is completely visible, or a game over event will be launched.
     * After this control, any full lines will be removed. If no full
     * lines could be removed, a figure start event is launched
     * directly.
     */
    private void handleFigureLanded() {
      if (!moveLock&&!moveLock2)
            return;

        // Check and detach figure
        if (figure.isAllVisible()&&figure2.isAllVisible()) {
            score += 10;
            handleScoreModification();

        } else {
            handleGameOver();
            return;
        }

        figure.detach();
        figure = null;
        figure2.detach();
        figure2 = null;



        // Check for full lines or create new figure
        if (board.hasFullLines()) {
            board.removeFullLines();
            if (level < 9 && board.getRemovedLines() / 20 > level) {
                level = board.getRemovedLines() / 20;
                handleLevelModification();
            }
        } else {
            handleFigureStart();
        }
    }

    /**
     * Handles a timer event. This will normally move the figure down
     * one step, but when a figure has landed or isn't ready other
     * events will be launched. This method is synchronized to avoid
     * race conditions with other asynchronous events (keyboard and
     * mouse).
     */
    private synchronized void handleTimer() {
        if ((figure == null)&&(figure2==null)) {
            handleFigureStart();
        } else if (figure.hasLanded()&&figure2.hasLanded()) {
            moveLock=true;
            moveLock2=true;
            handleFigureLanded();
        } else {
            figure.moveDown();
            figure2.moveDown();
        }

    }

    /**
     * Handles a button press event. This will launch different events
     * depending on the state of the game, as the button semantics
     * change as the game changes. This method is synchronized to
     * avoid race conditions with other asynchronous events (timer and
     * keyboard).
     */
    private synchronized void handleButtonPressed() {
        if (nextFigure == null&&nextFigure2==null) {
            handleStart();
        } else if (thread.isPaused()) {
            handleResume();
        } else {
            handlePause();
        }
    }

    /**
     * Handles a keyboard event. This will result in different actions
     * being taken, depending on the key pressed. In some cases, other
     * events will be launched. This method is synchronized to avoid
     * race conditions with other asynchronous events (timer and
     * mouse).
     *
     * @param e         the key event
     */
    private synchronized void handleKeyEvent(KeyEvent e) {


        // Handle start, pause and resume
        if (e.getKeyCode() == KeyEvent.VK_P) {
            handleButtonPressed();
            return;
        }

        // Don't proceed if stopped or paused
        if (figure == null || figure2 == null || (moveLock&&moveLock2)
|| thread.isPaused()) {
            return;
        }

        // Handle remaining key events
        switch (e.getKeyCode()) {


        case KeyEvent.VK_LEFT:
            if (!moveLock)
            figure.moveLeft();
            break;

        case KeyEvent.VK_RIGHT:
            if (!moveLock)
            figure.moveRight();
            break;

        case KeyEvent.VK_DOWN:
            if (!moveLock)
            figure.moveAllWayDown();
            moveLock = true;
            break;

        case KeyEvent.VK_UP:
        case KeyEvent.VK_SPACE:
            if (!moveLock){
            if (e.isControlDown()) {
                figure.rotateRandom();
            } else if (e.isShiftDown()) {
                figure.rotateClockwise();
            } else {
                figure.rotateCounterClockwise();
            }
            }break;

            //figure2
        case KeyEvent.VK_A:
            if (!moveLock2)
            figure2.moveLeft();
            break;


        case KeyEvent.VK_D:
            if (!moveLock2)
            figure2.moveRight();
            break;

        case KeyEvent.VK_W:
            if (!moveLock2)
            figure2.rotateCounterClockwise();
            break;

        case KeyEvent.VK_S:
            if (!moveLock2)
            figure2.moveAllWayDown();
            moveLock2 = true;
            break;

            //Controls
        case KeyEvent.VK_T:
            if (level < 9) {
                level++;
                handleLevelModification();
            }
            break;

        case KeyEvent.VK_N:
            preview = !preview;
            if (preview && figure != nextFigure) {
                nextFigure.attach(previewBoard, true,false);
                nextFigure.detach();
                nextFigure2.attach(previewBoard2, true,false);
                nextFigure2.detach();
            } else {
                previewBoard.clear();
                previewBoard2.clear();
            }
            break;
        }
    }

    /**
     * Returns a random figure. The figures come from the figures
     * array, and will not be initialized.
     *
     * @return a random figure
     */
    private Figure randomFigure() {
      return figures[(int) ( Math.random() * figures.length)];
    }
    private Figure randomFigure2() {
      return figures2[(int) ( Math.random() * figures2.length)];
    }


    /**
     * The game time thread. This thread makes sure that the timer
     * events are launched appropriately, making the current figure
     * fall. This thread can be reused across games, but should be set
     * to paused state when no game is running.
     */
    private class GameThread extends Thread {

        /**
         * The game pause flag. This flag is set to true while the
         * game should pause.
         */
        private boolean paused = true;

        /**
         * The number of milliseconds to sleep before each automatic
         * move. This number will be lowered as the game progresses.
         */
        private int sleepTime = 500;

        /**
         * Creates a new game thread with default values.
         */
        public GameThread() {
        }

        /**
         * Resets the game thread. This will adjust the speed and
         * start the game thread if not previously started.
         */
        public void reset() {
            adjustSpeed();
            setPaused(false);
            if (!isAlive()) {
                this.start();
            }
        }

        /**
         * Checks if the thread is paused.
         *
         * @return true if the thread is paused, or
         *         false otherwise
         */
        public boolean isPaused() {
            return paused;
        }

        /**
         * Sets the thread pause flag.
         *
         * @param paused     the new paused flag value
         */
        public void setPaused(boolean paused) {
            this.paused = paused;
        }
        /**
         * Adjusts the game speed according to the current level. The
         * sleeping time is calculated with a function making larger
         * steps initially an smaller as the level increases. A level
         * above ten (10) doesn't have any further effect.
         */
        public void adjustSpeed() {
            sleepTime = 4500 / (level + 5) - 250;
            if (sleepTime < 50) {
                sleepTime = 50;
            }
        }

        /**
         * Runs the game.
         */
        public void run() {
            while (thread == this) {
                // Make the time step
                handleTimer();

                // Sleep for some time
                try {
                    Thread.sleep(sleepTime);
                } catch (InterruptedException ignore) {
                    // Do nothing
                }

                // Sleep if paused
                while (paused && thread == this) {
                    try {
                        Thread.sleep(1000);
                    } catch (InterruptedException ignore) {
                        // Do nothing
                    }
                }
            }
        }
    }


    /**
     * A game panel component. Contains all the game components.
     */
    private class GamePanel extends Container {

        /**
         * The component size. If the component has been resized, that
         * will be detected when the paint method executes. If this
         * value is set to null, the component dimensions are unknown.
         */
        private Dimension  size = null;

        /**
         * The score label.
         */
        private Label scoreLabel = new Label("Score: 0");

        /**
         * The level label.
         */
        private Label levelLabel = new Label("Level: 1");

        /**
         * The generic button.
         */
        private Button button = new Button("Start");

        /**
         * Creates a new game panel. All the components will
         * also be added to the panel.
         */
        public GamePanel() {
            super();
            initComponents();
        }

        /**
         * Paints the game component. This method is overridden from
         * the default implementation in order to set the correct
         * background color.
         *
         * @param g     the graphics context to use
         */
        public void paint(Graphics g) {
            Rectangle  rect = g.getClipBounds();

            if (size == null || !size.equals(getSize())) {
                size = getSize();
                resizeComponents();
            }
            g.setColor(getBackground());
            g.fillRect(rect.x, rect.y, rect.width, rect.height);
            super.paint(g);
        }

        /**
         * Initializes all the components, and places them in
         * the panel.
         */
        private void initComponents() {
            GridBagConstraints  c;

            // Set layout manager and background
            setLayout(new GridBagLayout());
            setBackground(Configuration.getColor("background",
"#d4d0c8"));

            // Add game board
            c = new GridBagConstraints();
            c.gridx = 0;
            c.gridy = 0;
            c.gridheight = 5;
            c.weightx = 1.0;
            c.weighty = 1.0;
            c.fill = GridBagConstraints.BOTH;
            this.add(board.getComponent(), c);

            // Add next figure board
            c = new GridBagConstraints();
            c.gridx = 1;
            c.gridy = 0;
            c.weightx = 0.2;
            c.weighty = 0.18;
            c.fill = GridBagConstraints.BOTH;
            c.insets = new Insets(15, 15, 0, 15);
            this.add(previewBoard.getComponent(), c);

         // Add next figure board 2
            c = new GridBagConstraints();
            c.gridx = 1;
            c.gridy = 1;
            c.weightx = 0.2;
            c.weighty = 0.18;
            c.fill = GridBagConstraints.BOTH;
            c.insets = new Insets(15, 15, 0, 15);
            this.add(previewBoard2.getComponent(), c);

            // Add score label
            scoreLabel.setForeground(Configuration.getColor("label",

"#000000"));
            scoreLabel.setAlignment(Label.CENTER);
            c = new GridBagConstraints();
            c.gridx = 1;
            c.gridy = 2;
            c.weightx = 0.3;
            c.weighty = 0.05;
            c.anchor = GridBagConstraints.CENTER;
            c.fill = GridBagConstraints.BOTH;
            c.insets = new Insets(0, 15, 0, 15);
            this.add(scoreLabel, c);

            // Add level label
            levelLabel.setForeground(Configuration.getColor("label",

"#000000"));
            levelLabel.setAlignment(Label.CENTER);
            c = new GridBagConstraints();
            c.gridx = 1;
            c.gridy = 3;
            c.weightx = 0.3;
            c.weighty = 0.05;
            c.anchor = GridBagConstraints.CENTER;
            c.fill = GridBagConstraints.BOTH;
            c.insets = new Insets(0, 15, 0, 15);
            this.add(levelLabel, c);

            // Add generic button
            button.setBackground(Configuration.getColor("button",
"#d4d0c8"));
            c = new GridBagConstraints();
            c.gridx = 1;
            c.gridy = 4;
            c.weightx = 0.3;
            c.weighty = 1.0;
            c.anchor = GridBagConstraints.NORTH;
            c.fill = GridBagConstraints.HORIZONTAL;
            c.insets = new Insets(15, 15, 15, 15);
            this.add(button, c);

            // Add event handling
            enableEvents(KeyEvent.KEY_EVENT_MASK);
            this.addKeyListener(new KeyAdapter() {
                public void keyPressed(KeyEvent e) {
                    handleKeyEvent(e);
                }
            });
            button.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                    handleButtonPressed();
                    component.requestFocus();
                }
            });
        }

        /**
         * Resizes all the static components, and invalidates the
         * current layout.
         */
        private void resizeComponents() {
            Dimension  size = scoreLabel.getSize();
            Font       font;
            int        unitSize;

            // Calculate the unit size
            size = board.getComponent().getSize();
            size.width /= board.getBoardWidth();
            size.height /= board.getBoardHeight();
            if (size.width > size.height) {
                unitSize = size.height;
            } else {
                unitSize = size.width;
            }

            // Adjust font sizes
            font = new Font("SansSerif",
                            Font.BOLD,
                            3 + (int) (unitSize / 1.8));
            scoreLabel.setFont(font);
            levelLabel.setFont(font);
            font = new Font("SansSerif",
                            Font.PLAIN,
                            2 + unitSize / 2);
            button.setFont(font);

            // Invalidate layout
            scoreLabel.invalidate();
            levelLabel.invalidate();
            button.invalidate();
        }
    }


      public void onReceivedSignal(String command) {
              // Don't proceed if stopped or paused
        if (figure == null || figure2 == null || (moveLock&&moveLock2)
|| thread.isPaused()) {
            return;
        }
            if (command.equals("Left")){

                  figure.moveLeft();
            }
            else if(command.equals("Right")){

                  figure.moveRight();
            }
            else if(command.equals("Up")){

                  figure.rotateCounterClockwise();
            }
            else if(command.equals("Down")){

                  figure.moveAllWayDown();
                  moveLock = true;
            }
      }
}
```

IStuffProxy.java

```
package net.tetriscollaborative;
import java.util.ArrayList;
import java.util.EventListener;
import java.util.List;


import iwork.eheap2.EventHeap;
import iwork.eheap2.EventHeapException;


public class IstuffProxy implements Runnable{

      //Interface for EventListener
      public static interface ControlListener extends EventListener{
        void onReceivedSignal(String command);
    }
      private List<ControlListener> listeners=new
ArrayList<ControlListener>();
      iwork.eheap2.Event templateEvent;
    EventHeap m_EventHeap;

    IstuffProxy(String server)
    {
        super();
        System.out.println ("Trying to connect\n");
        m_EventHeap = new EventHeap(server);
        System.out.println ("Did we connect?");
        try {
            templateEvent = new iwork.eheap2.Event("ControlEvent");
        } catch(Exception ex) { ex.printStackTrace(); }
        //setDaemon(true);
    }
    //Methods to add/remove listeners
    public void addListener(ControlListener listener){
        listeners.add(listener);
    }
    public void removeListener(ControlListener listener){
        listeners.remove(listener);
    }
    //method launching event
    protected void onReceivedSignal(String command){
        for(ControlListener listener:listeners)
            listener.onReceivedSignal(command);
    }

    public void run()
    {

        while(true) {
            try {
                iwork.eheap2.Event e =
m_EventHeap.waitForEvent(templateEvent);
                handleEventHeapEvent(e);
            } catch(Exception ex) { ex.printStackTrace(); }
        }
    }

    public void handleEventHeapEvent(iwork.eheap2.Event e)
    {
      try {

            String comand=e.getPostValueString("ControlCommand");
                  System.out.println(comand);
                  onReceivedSignal(comand);

            } catch (EventHeapException e1) {
                  e1.printStackTrace();
            }

    }

}
```