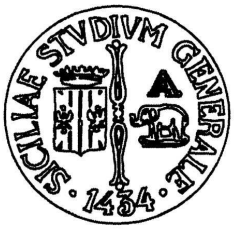
BachelorsThesis: Real World and Virtual World Architecture for Interconnecting First and Second Life

Filippo Sanfilippo, University of Catania, Italy



UNIVERSITÀ DEGLI STUDI DI CATANIA
FACOLTÀ DI INGEGNERIA
CORSO DI LAURE IN INGEGNERIA INFORMATICA

FILIPPO SANFILIPPO

REAL WORLD AND VIRTUAL WORLD ARCHITECTURE FOR INTERCONNECTING FIRST AND SECOND LIFE

TESI DI LAUREA

Relatrice:
Chiar.ma Prof.ssa A. Di Stefano
Correlatore:
Chiar.mo Ing. G. Morana

ANNO ACCADEMICO 2007/2008

I want to dedicate my thesis to my dear parents for all they have done for me and for each their sacrifice, to my sister and my brother and to my dearly beloved girlfriend.

I thank to all my friends that have been close.

I am very grateful to Michael Callaghan, lecturer of School of Computing and Intelligent Systems, University of Ulster.

Contents

Abstract

This thesis researches integrating the real world with virtual worlds where real world data is imported, manipulated and visualized in a virtual world. The overall objective is to create a generic architecture for interconnecting the real and virtual worlds with the end purpose of establishing real-time communications between them. In recent years the seemingly ever increasing computational resources of computer systems facilitate the development of new, better ways of data visualization and interaction between users and computers. One of the most powerful mechanisms of data visualization and interaction is Virtual Reality. Research teams at companies like Amazon, Nature and IBM are building new 3D environments in Second Life to investigate possible uses for these new virtual worlds. Applications like cellular tomography, Flickr photos and Wimbledon sports telemetry are just some of the services and devices people are using to blur the boundaries between their online and offline lives. The ability to link real and virtual worlds is an increasingly important area of research.

The focus of this project is Second Life. I chose this virtual world as it is extremely flexible and open to development. The Second Life client is open source and accessing the virtual world is free. Second Life provides inworld tools which are highly intuitive and allow the creation and manipulation of geometric primitives in a simple interface making it really easy to recreate real objects inside this virtual world. In addition one of the most important aspects of Second Life is the fact that every prim (building block object) can contain executable code script which can control the appearance and behaviour of the prim allowing it to communicate with avatars and with other prims while responding to events. This means that thanks to Second Life you can easily build any object and decide its behaviour by a simple script. The interconnection between the two worlds is made by associating a real event happening in the real world to its virtual equivalent allowing a real event to trigger a virtual one and vice-versa.

The challenge for this project is to create a real, permanent link between a real object and the virtual representation of it so that if something happens to the real object it also happens to the virtual one, and vice-versa. The approach offered in this project is a first attempt at solving this problem and designing a system to achieve this objective. To ensure my project is of value to a wider audience I have proposed and build an architecture that is as “hardware independent” as possible allowing an “easy plug-in” approach which is open to future developments.

The rationale for doing this is to create a generic architecture where it is possible to realize a lot of different applications starting from the simple remote control of the light or of the air conditioning in your house, to the use of virtual instrumentation/remote experimentation / virtual learning and more generally all domestic area’s and eventually teleworking... or simply to shorten the distance between two worlds.

Chapter 1
Introduction

1.1 From my training to my thesis

To understand the thesis work, it is important to understand how it came about.
I applied for and received a place on the “Leonardo Da Vinci” program. This project which is promoted by the European Union helps students to undertake a period of study in another European country. Under this scheme I spent September, 24th 2007 to January, 22nd 2008 on a training placement as a researcher at the Intelligent Systems Research Centre, University of Ulster (Northern Ireland).

During my traineeship, I have worked with my colleague Giorgio Scibilia in a project titled "Real World Sensors and Second Life: Interconnecting first and second life". This project concerns the integration between hardware and the virtual reality world in Second Life and it has led to the creation of a designed space in the virtual world using physical sensors data as input. The project focuses on the basic concepts of the connection between the real world and the virtual world.

Initially, the idea was to interconnect specific hardware to the virtual world. We wanted to use the "EM2420 evaluation kit“. This kit is a network of sensors that uses the ZigBee 802.15.4 standard protocol. But the most problem we have encountered is that The “Ember Studio Lite” does not permit to store or simply export the data taken from the sensors so we have had to find a different solution to this problem. After some consideration I proposed an architectural scheme independent from hardware which was flexible and open to possible future developments/enhancements. This approach and the architecture proposed was much appreciated by my supervisor and our project took a decisive turning point.

As a next step we developed the basic architecture for realizing a representative 1:1 connection e.g. if a light source is switched on in the real world, its equivalent is switched on in the virtual world and then we created its physical manifestations through a practical demo. Then, using and extending on the existing architecture we have worked on the interconnection of test and measurement instruments (waveform generator, multimeter, oscilloscope) with Second Life.

I think that my internship has been a very important formative experience for my career and also for my life in general. I have learned to address problems regarding reverse engineering and software development, working to extraordinarily high levels. During the design and development phase, I have had to face new problems that did not have solutions before. At the end of my traineeship I was able to write a comprehensive conference paper documenting my work.

From this project we subsequently developed a part focused on e-learning where the idea was to use our pre-existent architecture for implementing and evaluating innovative pedagogical approaches to teaching in 3d virtual worlds without compromising the cultivation of traditional skills. This part of the project explores the use of Second Life to create experiential based learning experiences in a 3D immersive world for teaching the principles of computer hardware architectures and electronic circuits.

It describes the functionality available in Second Life to model, capture, display and visualize real world data and explores the barriers to controlling real world instrumentation through existing software packages. About this we have wrote a paper titled "Experiential based learning in 3D Virtual Worlds: Visualization and data integration in Second Life" (Callaghan M.J., Harkin J., Scibilia G., Sanfilippo F., McCusker K., Wilson S.). The paper won the “best paper award” at REV Conference 2008 organized by the University of Applied Sciences in Duesseldorf (Germany, June 2008).

Thanks to my experiences I have decided to write a thesis about the possibility of interconnecting between real and virtual world.

1.2 Scenario

The concept of online virtual worlds where a user can interact with other people and fantastic landscapes has fascinated hackers and science fiction writers for years.
Today, the traditional media, such as the print, film, radio and television, co-exist with the new interactive media (the Internet being the leading one), which are not only opening up possibilities of global information saving, real-time data transfer and of on-line communication, but also have numerous anthropological, aesthetical, cultural, social and political implications. These media are also a means of forming new cultural contents and means of establishing manners and approaches of how these contents are accepted, which has quite an impact on contemporary art, on life styles, fashion and design. Internet culture is an integral part of a wider concept of cyberculture, its essential concepts being interactivity, (total) immersion and participation as role and identity switching, dematerialization of the object and decentralization of the subject.

An always increasing number of people want to be present on internet as an avatar. In other words, virtual reality has broken the limits of diffusion and culture for becoming a mass phenomenon, a new channel of communication, management and production of knowledge, or rather, production of experiences. For this reasons, the digital space opens new scenarios that need new languages, metaphors and solutions to cope with opportunities, items and problems that can not be put on the Internet in the form of pages.
The spread of virtual world is due in large part to the emotional and expressive strength and to the possibility of representing an alter-ego, a world whit objects and behaviors. The virtual world has become an extension of the real world. I think that currently the peculiarities of virtual world is not in the effectiveness of management information and knowledge, which remains for now more linked to web pages, but in its the emotional and expressive power.

1.3 Virtual world

A virtual world [1] is an interactive simulated environment accessed by multiple users through an online interface. Virtual worlds are also called "digital worlds," "simulated worlds" and "MMOG's."[1]. There are many different types of virtual worlds, however there are some features all of them have in common:

- Shared Space: the world allows many users to participate at once.
- Graphical User Interface: the world depicts space visually, ranging in style from 2D "cartoon" imagery to more immersive 3D environments.
- Immediacy: interaction takes place in real time.
- Interactivity: the world allows users to alter, develop, build, or submit customized content.
- Persistence: the world's existence continues regardless of whether individual users are logged in.
- Socialization/Community: the world allows and encourages the formation of in-world social groups like teams, guilds, clubs, cliques, housemates, neighbourhoods, etc.

Virtual worlds have been created for many different purposes. The largest and most common type of virtual world is the "MMORPG" which stands for "Massively Multiplayer Online Role Playing Game." But virtual worlds have also been built for purposes other than gaming as socializing and education.
In general a computer/user can access a computer-simulated world and presents perceptual stimuli to the user, who in turn can manipulate elements of the modeled world and thus experiences telepresence to a certain degree. Such modeled worlds may appear similar to the real world or instead depict fantasy worlds. The model world may simulate rules based on the real world or some hybrid fantasy world. Example rules are gravity, topography, locomotion, real-time actions, and communication. Communication between users has ranged from text, graphical icons, visual gesture, sound, and rarely, forms using touch and balance senses.
Massively multiplayer online games commonly depict a world similar to the real world, with real world rules and real-time actions, and communication. Communication is usually textual, with real-time voice communication using VOIP also possible.
Virtual worlds are the most important and tangible demonstration of how much is the general desire to have a parallel life, in every aspect comparable to the real one.

Chapter 2
Virtual world choice: Second Life

2.1 What is Second Life

Figure 2.1: Second Life logo

Second Life, established in 2002 by San Francisco based software company Linden Labs, is an alternative online game currently hyped by the media. Whether you call Second Life a game, social networking tool or a programming environment depends on how you use it [2].

Second Life is based on fully customizable avatars evolving in a virtual 3D space and incorporates a variety social networking tools.

Second Life's virtual world also includes sound; wind in the swaying trees, babbling brooks, audible conversation, and built-in chat and instant messaging. Residents buy property, start businesses, game with other residents, create objects, join clubs, attend classes, or just hang out. The rendering of Second Life is remarkable, making it an instant guilty pleasure.

Since opening to the public in 2003, it has grown explosively and today is inhabited by millions of Residents from around the globe.

From the moment you enter the World you'll discover a vast digital continent, teeming with people, entertainment, experiences and opportunity. Once you've explored a bit, perhaps you'll find a perfect parcel of land to build your house or business.

You'll also be surrounded by the Creations of your fellow Residents. Because Residents retain intellectual property rights in their digital creations, they can buy, sell and trade with other Residents.

The Marketplace currently supports millions of US dollars in monthly transactions. This commerce is handled with the inworld unit of trade, the Linden™ dollar, which can be converted to US dollars at several thriving online Linden dollar exchanges.

2.2 Some technical information

The flat, Earth-like world of Second Life is simulated on a large array of Debian servers, referred to as the Grid [3].

The Second Life Grid is the platform and technology behind 3D online virtual world Second Life.

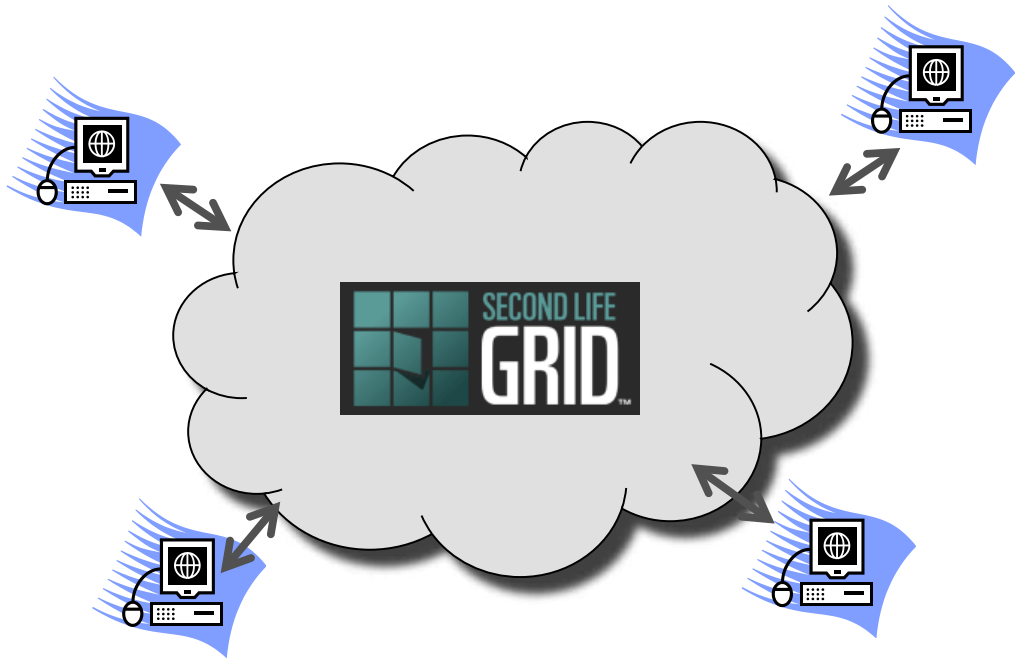


Figure 2.2: Second Life Grid

The world is divided into 256x256 m areas of land, called Regions. Each Region is simulated by a single named server instance, and is given a unique name and content rating (either PG or Mature).

Multiple server instances can be run on a single physical server, but generally each instance is given a dedicated CPU core of its own. Modern servers with two dual-core processors usually support four separate server instances.

The Second Life world runs on Linden Time, which is identical to the Pacific Time Zone. The virtual world follows the North American Daylight Saving Time convention. Hence it runs 7 hours behind UTC most of the year, and 8 hours behind when Standard Time is in effect during the winter. The servers' log files actually record events in UTC, however.

• Physics simulation

Each server instance runs a physics simulation to manage the collisions and interactions of all objects in that region. Objects can be nonphysical and nonmoving, or actively physical and movable. Complex shapes may be linked together in groups of up to 255 separate primitives. Additionally, each player's avatar is treated as a physical object so that it may interact with physical objects in the world.

As of April 1, 2008, Second Life simulators use the *Havok 4* physics engine for all in-game dynamics. This new engine is capable of simulating thousands of physical objects at once. However, more than 500 constantly interacting collisions have noticeable impact on simulator performance. The previous Havok 1 installment of the physics engine caused what is known as the *Deep Think* condition; processing overlapping object collisions endlessly. It has been alleviated through the introduction of an *overlap ejection* capability. This allows overlapped objects to separate and propel apart as if compressing two springs against each other.

• Asset storage

Every item in the Second Life universe is referred to as an *asset*. This includes the shapes of the 3D objects known as *primitives*, the digital images referred to as *textures* that decorate primitives, digitized audio clips, avatar shape and appearance, avatar skin textures, LSL scripts, information written on notecards, and so on. Each asset is referenced with a universally unique identifier or *UUID*.

Assets are stored in their own dedicated MySQL server farm, comprising all data that has ever been created by anyone who has been in the SL world.

As the popularity of Second Life has increased, the strain on the database engine to quickly and efficiently store and retrieve data has also continued to increase, frequently outpacing the ability of the Linden staff to keep their asset farm equipped to handle the number of users logged into the world at the same time.

Under severe load conditions it is common for the database engine to simply not reply to requests in a timely fashion, causing objects to not rez or delete as expected, or for the client inventory to not load, or the currency balance to not appear in the client program. Searching for locations, people, or classifieds may also fail under heavy load conditions. The database load is typically the most severe on weekends, particularly Sunday afternoons (Second Life Time), while the system can function just fine when accessed during low-load times such as at night or in the middle of the week during the day.

- ## Software

The Second Life software comprises the *viewer* (also known as the *client*) executing on the Resident's computer, and several thousand servers operated by Linden Lab. There is an active *beta-grid* that has its own special client, which is updated very regularly, and is used for constant software testing by volunteers. This testing software was introduced to eliminate the short amounts of time between real updates, and increase its overall quality. The beta-grid reflects the standard main-grid, except that the actions taken within it are not stored by the servers; it is for testing purposes only. Every few months, the standard software is replaced by the beta-grid software, intended as a big upgrade.

The Second Life user-base is growing rapidly, and this has stimulated both social and technological changes to the world; the addition of new features also provides periodic boosts to the growth of the economy.

Linden Lab pursues the use of open standards technologies, and uses free and open source software such as Apache, MySQL and Squid. The plan is to move everything to open standards by standardizing the Second Life protocol.

Linden Lab provides viewers for Microsoft Windows 2000/XP, Mac OS X, and most distributions of Linux. As of mid-2007, Microsoft Windows Vista is not yet officially supported although the viewer will generally run on Vista systems. In the past, viewer upgrades were usually mandatory; the old viewer would not work with the new version of the server software. However, Linden Lab is working on a more flexible protocol that will allow clients and servers to send and take whatever data they may require, hence differing versions would nonetheless be able to work together. The project is known as Het-Grid or heterogeneous grid and the first iteration of the server software was deployed to the Main Grid over a few weeks in August 2007.

As of January 8, 2007, the Viewer is distributed under version 2 of the GNU General Public License, with an additional clause allowing combination with certain other free software packages which have otherwise-incompatible licenses. Currently not all of the required dependencies have been released.

2.3 The Second World

The most important thing about the Second Life world is that it is constantly changing and growing [4]. Here's why:

- Thousands of new residents join each day and Create an Avatar
- Those avatars Explore the World and Meet People
- Some people decide to purchase Virtual Land, which allows them to open a business, build their own virtual paradise, and more
- Linden Lab creates new land to keep up with demand. What began as 64 acres in 2003 is now over 65,000 acres and growing rapidly.

• Creating an avatar

The word avatar derives from the Sanskrit "avatara" meaning "incarnation." For Second Life members, it is a means of reinventing oneself by constructing a 3-D animated model. Once you have chosen an avatar, you can customise with it as you like.

The Second Life world is a place dedicated to your creativity.

After all, an avatar is your persona in the virtual world. The picture below shows how easy it is to create your avatar. Despite offering almost infinite possibilities, the tool to personalize your avatar is very simple to use and allows you to change anything you like, from the tip of your nose to the tint of your skin. Don't worry if it's not perfect at first, you can change your look at any time.

Figure 2.3: Second Life tool for personalizing your avatar

• Exploring

When a place is as big as the Second Life world and has so many things to do, the first question people ask is, "Where do I start?".
There are three tools to help you start exploring. As an example, let's pretend you're looking for a good place to party and meet people.

1. **The Map.** Imagine you want to find people socializing. View upcoming events' information on the web map by clicking the green dots being shown.

2. **The Search Menu.** Now you'd like to learn more about a particular event you saw on the web map. Search for it, and read its description. It sounds great, and you teleport there.

3. **Other People.** Once you arrive, right-click on anybody's avatar and choose "View Profile" to learn about them.

Figure 2.4: Second Life map, search menu, avatar's profile

- **Meeting people**

One reason that people join second life is for the social aspect of this game. Basically this is a great way to meet new people; you can find people from various countries playing this game. And along with meeting new people comes another major aspect of this game which is communicating with other avatars. There are numerous people that you can meet throughout the game and most people are very eager to meet you. In fact because communication is an important aspect in the virtual world there are various ways that you can communicate with other avatars.

Within this vibrant society of people, it's easy to find people with similar interests to you. Once you meet people you like, you find it's easy to communicate and stay in touch.

At any time there are dozens of events where you can party at nightclubs, attend fashion shows and art openings or just play games.

Residents also form groups ranging from neighbourhood associations to fans of Sci Fi Movies.

- **Owning virtual land**

Owning land in the Second Life world allows you to build, display, and store your virtual creations, as well as host events and businesses. But to do that, you need to acquire some virtual land.

The Pricing and Fees are simple; you pay $9.95 a month plus a Land Use Fee proportional to the amount of land you own.

Whether it's a modest nook for a relaxing cottage, or an entire island to build your dream amusement park, land is for everyone.

There are several ways to acquire a piece of land in Second Life. Here are the most important ones.

When new land is available in the Second Life World, it could be posted in an auction system. The mechanic of the auction system is similar to common platforms like eBay. The bit can be made in Linden Dollars or US Dollars.

Figure 2.5: Available lands in Second Life

2.4 Creating content

The Second Life world is a place dedicated to your creativity. It's about dreaming of something one moment and bringing it to life the next. Everything in Second Life is resident-created, from to the strobe lights in the nightclubs to the car (or spaceship) in your driveway.

Imagine tinkering with the steering and handling program of a motorcycle while your friend tweaks the shape of the fuel tank and gives it a wicked flame paint job, in-world and in real-time, before you both take it for a spin down a newly created road to look for some land to buy. Have you ever wondered what it would be like to have a pair of black leather wings? Build them and give it a go.

You can:

- See what residents are building right now
- Browse the thousands of items for sale on community websites
- See how residents are pushing the envelope making movies with Second Life
- Learn more about how to Create Anything in Second Life

Figure 2.6: Tools for creating objects in Second Life, editing

- **Create anything**

In the Second Life world you can create anything you can imagine with powerful, highly flexible building tools, using geometric primitives and a simple, intuitive interface. Build live, in real-time, right in Second Life no separate tools or applications to buy or learn and no hassles with importing your work. Building is easy to learn, yet robust enough to inspire your creativity.

A powerful but fun-to-learn and use scripting language further enhances your creations by allowing you to add behaviours to the objects you build sculpt a butterfly, then write a short chunk of code that lets it follow you around as you walk.

And once you've built something, you can easily begin selling it to other residents, because you control the IP Rights of your creations.

What if you want something but don't quite have the time or skills to make it? Just do a quick search to find and buy what you want.

Figure 2.7: Tools for creating objects in Second Life, setting colours

• Building

Internal Building System

Highly flexible building tools allow manipulation of geometric primitives in a simple interface. You can stretch these “prims” into new shapes, change their texture and physical qualities, link them to other prims. From this easy-to-learn process, it is possible to create objects of all kinds and sizes, from a jewelled navel ring to a five hundred meter skyscraper.

Importable Textures

You can also import .jpegs and other standard graphic files into the Second Life world to texture the objects you create.

Collaborative Creation

It is possible to construct the same object(s) with several other Residents (at different times, or simultaneously).

Physics and Dynamic Lighting

You can imbue all objects with Havok™ powered physics so they respond to gravity, inertia, propulsion, and wind from the in-world weather system. Objects cast shadows and are illuminated by the sun.

• Scripting

One of the most important aspects of Second Life is the fact that every prim (building block object) can contain executable code script which can control the appearance and behavior of the prim, communicate with avatars and with other prim, respond to events, and so forth. The scripting language is called LSL (Linden Scripting Language) and is documented in the LSL Wiki.

LSL is the language all scripts in Second Life are written in. A script in Second Life is a set of instructions that can be placed inside any primitive object in the world, but not inside an avatar. Avatars, however, can wear scripted objects.

Multiple scripts may also be attached to the same object, allowing a style of small, single-function scripts to evolve. This leads to scripts that perform specific functions ("hover", "follow", etc.) and allows them to be combined to form new behaviours.

The text of the script is compiled into an executable byte code, much like Java. This byte code is then run within a virtual machine on the simulator. Each script receives a time slice of the total simulator time allocated to scripts, so a simulator with many scripts would allow each individual script less time rather than degrading its own performance. In addition, each script executes within its own chunk of memory, preventing scripts from writing into protected simulator memory or into other scripts, making it much harder for scripts to crash the simulator.

This language follows the familiar syntax of a c/Java style language, with an implicit state machine for every script. Mainly it has a large number of built-in functions and event handlers specific to the 3D virtual environment and the manipulation of user-created objects. In addition to the string and numeric variable types, there are variable types specific to location and movement in the 3D world which provide the basis for dramatic animation effects. Collections are limited to a list type which can hold any of the other variable types.

One thing that makes LSL special is its emphasis on "States" and "Events". A door can be "open" or "closed" and a light can be "on" or "off". A person can be "hyper", "calm", or "bored". Many real life behaviours can be modelled with "states" and the same can be true for LSL programs. Minimally a script will have one state, the default state.

When a script is first started or reset, it will start out in the default state.

The default state is declared by placing the default at the root level of the document, and marking the beginning with an open brace '{' and ending with a close brace '}'. Because of it's privileged status, you do not declare that it is fact a state like you normally would with other states.

Every time you enter a state, the script engine will automatically call the state_entry() event and execute the code found there. On state exit, the script engine will automatically call the state_exit() event before calling the next state's state_entry handler.

An event can be thought of as a "Trigger". Events are not user defined in Second Life, but rather predefined. They are either caused by objects and avatars interacting in the world, or they are created in a script. Events trigger event handlers (sometimes just called "events" as well). For example, when an avatar touches an object, a *touch_start* message is sent to the object, which causes the touch_start() event handler to begin executing. So the minimum LSL program must have one state with one event handler in it. Here is a look at a minimal program written in LSL that can loosely be translated as...."When I am in the default state, and I am touched, say "Hello World" on channel zero".

```
default
{
      touch_start(integer total_number)
      {
            llSay(0,"Hello World");
      }
}
```

Listing 2.1: A minimal program written in LSL

Scripts can make an object move, listen, talk, operate as a vehicle or weapon, change color, size or shape. A script can make an object listen to your words as well as talk back to you, scripts even let objects talk to each other.

The most basic object in Second Life is the "Prim" or primitive, the basic building block of all objects you can build in Second Life. When several prims are linked, they can each contain a script which speaks to the rest of the object via Link Messages. These are faster and more private than having objects "chat" or email each other. These are beyond the scope of this tutorial and we will instead focus on single scripts in a single prim.

Scripting is harder to learn than basic object manipulation, but is very rewarding once you make progress.

If you have built in Second Life, everything you can define in the edit window can be defined in a script. All interaction you see between objects or between avatars and objects is via scripts.

Learning more about the world and building model is vital to some aspects of scripting, thus I would recommend a good foundation in building as you learn to script.

2.5 The Marketplace

In Second life it is also possible to make real money. That's right, real money.

Here's how it works:

- The Second Life world has a fully-integrated economy architected to reward risk, innovation, and craftsmanship.
- Residents create their own virtual goods and services. Because residents retain the IP rights of their creations, they are able to sell them at various in-world venues.
- Businesses succeed by the ingenuity, artistic ability, entrepreneurial acumen, and good reputation of their owners.
- Residents who have amassed lots of Linden dollars are matched with residents who want to buy Linden dollars at the Linden exchange (the official Linden dollar exchange), or at other unaffiliated third-party exchanges.
- The Second Life real estate market provides opportunities for Residents to establish their own communities and business locations.

Figure 2.8: A shop in Second Life

Economy

There is an Internal Unit of Trade with Real Market Value

Millions of Linden dollars change hands every month for the goods and services Residents create and provide. This unit of trade may then be bought and sold on the Linden exchange (the official Linden dollar exchange of the Second Life world), or other unaffiliated third party sites for real currency.

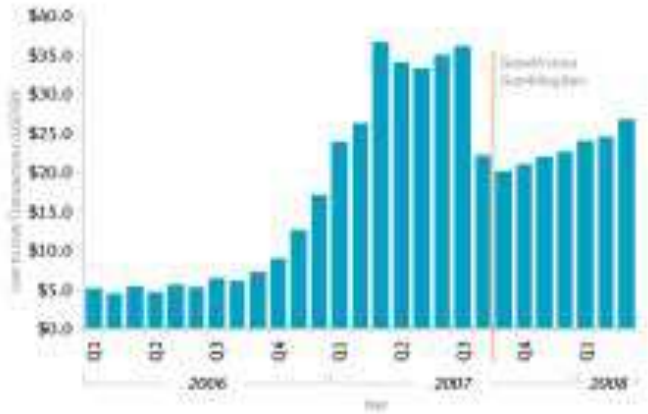


Figure 2.9: User to User Transactions

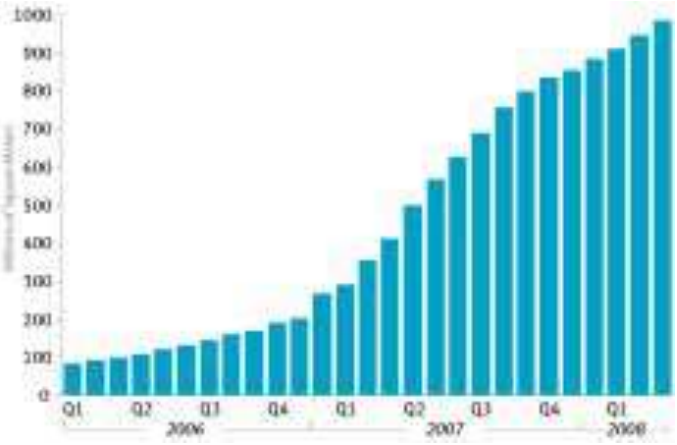


Figure 2.10: Resident Owned Land Mass

Business opportunities

There are as many opportunities for innovation and profit in the Second Life world as in the Real World. Open a nightclub, sell jewelry, become a land speculator; the choice is yours to make. Thousands of residents are making part or all of their real life income from their Second Life Businesses.

By way of example, here are just a few in-world business occupations which Residents founded and currently run, and make part or all of their real life living from. Try searching for these categories via the in-world Find menu:

- party and wedding planner
- pet manufacturer
- tattooist
- nightclub owner
- automotive manufacturer
- fashion designer
- aerospace engineer
- custom avatar designer
- jewelry maker
- architect
- XML coder
- freelance scripter
- game developer
- fine artist
- machinima set designer
- tour guide
- musician
- custom animation creator
- theme park developer
- real estate speculator
- vacation resort owner
- advertiser
- bodyguard
- magazine
- publisher
- private detective
- writer
- gamer
- landscaper
- publicist
- special effects designer
- gunsmith
- hug maker

2.6 Why Second Life

I have chosen Second Life because I think that it is quite more than a virtual playground. It is an extremely flexible virtual world and open to development. The client is now open source and importantly it is free to play. Moreover the Second Life client is equipped with highly intuitive building tools allowing the creation and manipulation of geometric primitives in a simple interface. So it is really easy to recreate real objects inside the virtual world of Second life.

But one of the most important aspects of Second Life is the fact that every prim (building block object) can contain executable code script. With proper scripting, objects can interact with avatars and other objects. Objects can detect the presence of nearby avatars, "hear" chat that is "spoken" by users and respond to being "touched." Objects can also use the connectivity capabilities to exchange data with the outside world. In short, thanks to Second Life you can easily build any object and decide its behaviour by a simple script.

Besides there are now so many businesses experimenting with a Second Life presence that I can't begin to list them all. If you read the trade press for practically any industry you have been seeing news articles. I just found out there is a "Second Life Corporate Business Council" with 30-40 big business members! In addition to the obvious display of advertising banners with links to corporate Web sites, creating environments for virtual conversations with customers has been found valuable by Dell and IBM, among other big business applications.

In my opinion, Second Life is the first 3D virtual world with enough functionality to justify serious experimentation with commercial possibilities. Experience gained in Second Life will probably show up in private corporate virtual worlds.

Chapter 3
Connecting the real world and Second Life

3.1 Background

The integration of real world hardware devices within Second Life represents a significant challenge for a lot of reasons. Firstly, Second Life was not designed or developed with hardware integration in mind and therefore no direct mechanisms or Application Programming Interfaces (APIs) exist that support this endeavor. Secondly, the development of a custom solution to hardware integration depends on the direct transmission of raw data into and out of the Second Life world. Once again, this was not a design goal of Second Life and the mechanisms in place to support data transmission between the real world and Second Life are currently limited.

There are many people and also companies (Serious Games Institute and also IBM have projects about that) interested in linking real and second life, and someone was already able to link external hardware to SL. Some of these projects are now discussed.

Link 1: http://www.hackdiary.com/archives/000101.html

This blog contain a description of a working prototype for connecting the real world and Second Life using an Arduino hardware prototyping board. Analogue values were read from a potentiometer and feedback via the USB-serial interface to a Mac and onto Second Life using a modified version of the viewer client. The results demo allows the user to control and manipulate a 3D object in the virtual world with approximately a 1 second lag or time delay between stimulus and response.

Figure 3.1: Arduino hardware moving a box inside Second Life

Link 2:
http://www.ibr.cs.tu-bs.de/theses/roehr/pa-realworldsensornodesandsecondlife.html?lang=en

A german student is preparing a thesis about “Real World Sensor Nodes and Second Life: Interconnecting first and second life”.

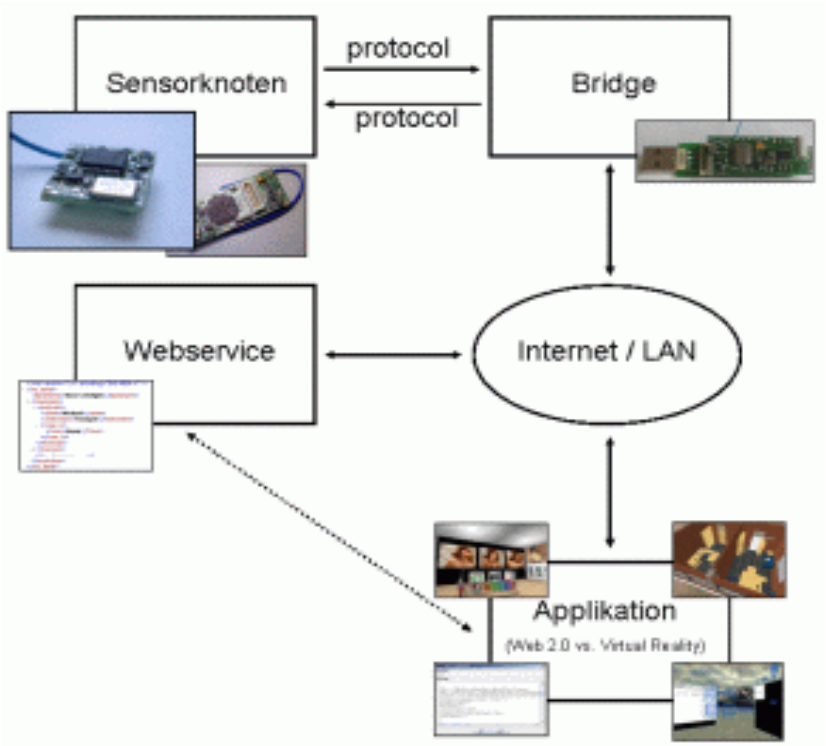


Figure 3.2: Possible way for interconnecting first and SL, from a student thesis

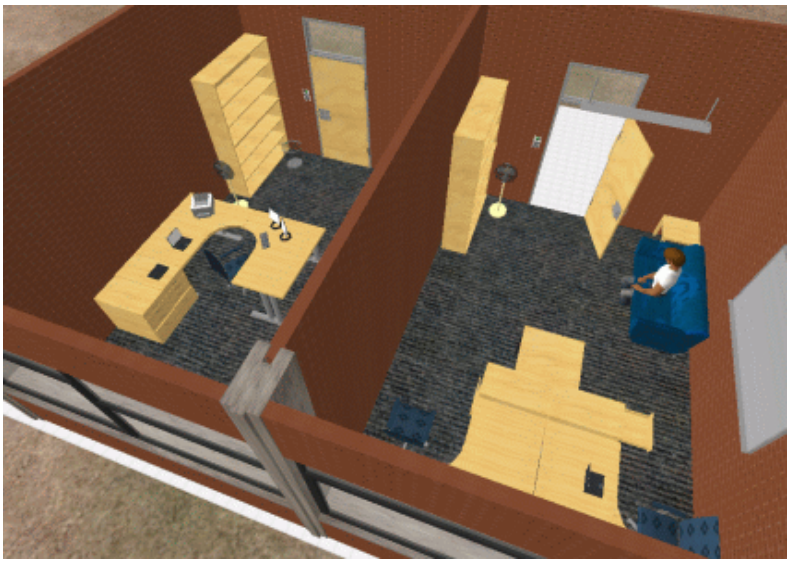


Figure 3.3: Possible application in home automation, from a German student thesis

Link 3: http://www.media.mit.edu/resenv/pubs/papers/2007_04_lifton_ipsn.pdf

This pdf shows the connection between the "plug" (a plug sensor network) and second life

Link 4: http://www.secondlifeinsider.com/2006/10/28/3d-weather-data-visualization-in-second-life/

Website about "3D Weather Data Visualization in Second Life", a weather simulator rendered in Second Life.
The system works by way of dozens of scripted reporting stations dotted all over a map of the United States. These stations retrieve METAR data from NOAA every eight minutes which they then decode and render into models of the appropriate weather phenomenon for the area. All sorts of cloud cover and precipitation models are available as well as special weather conditions such as thunderstorms and tornadoes. Temperature is represented by warmer and cooler shades of color. This 3D composite is great for giving visitors a visceral feel for the weather around them.

Figure 3.4: 3D Weather Data Visualization in Second Life

Link 5:http://www.secondlifeinsider.com/2006/10/28/3d-weather-data-visualization-in-second-life/ . It's a blog that explains how SL and a MYSQL database can communicate.

3.2 High level model

The high level view allows us to define the system elements and how they interact. At this stage we can suppress purely local information about the single elements for focusing our attention on the architectural scheme. The aim, practically, is to take data from the real world and bring them to Second Life.

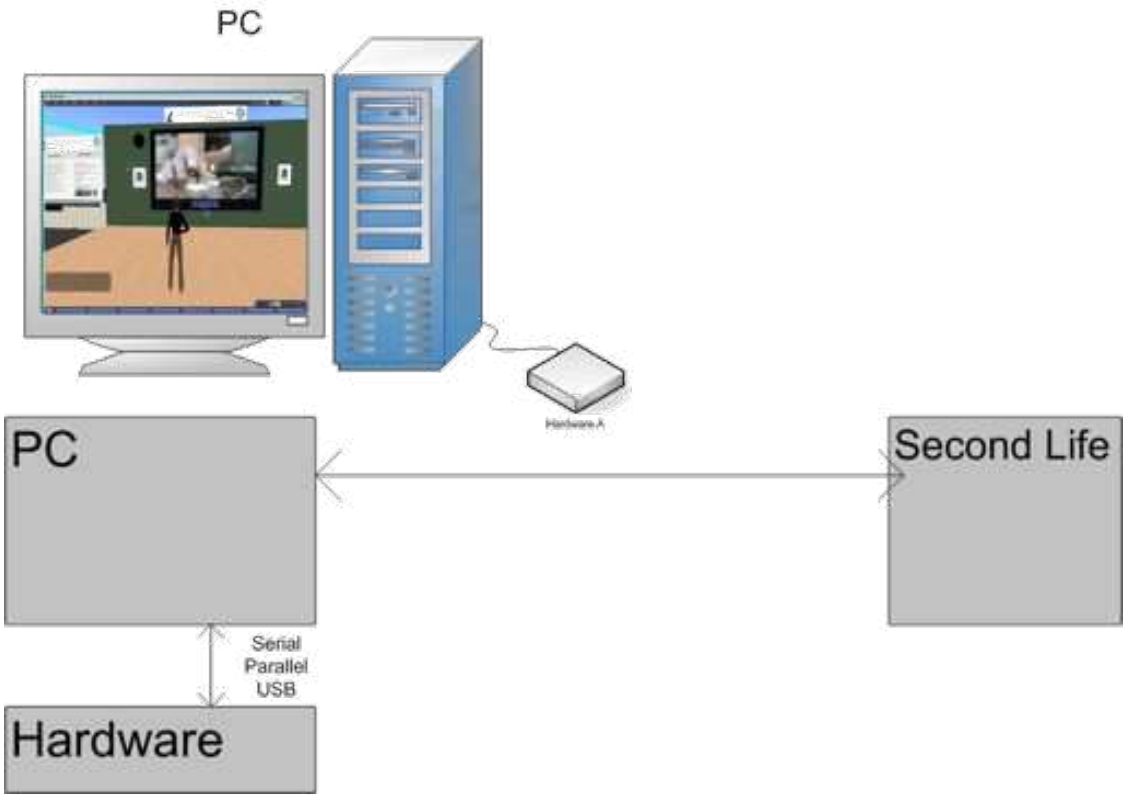


Figure 3.5: High level model

Basically, the main problem is to find the right way of sending data to SL, and to receive from it too. So, the first problem to solve, is finding out “from where” SL can send/receive data.

3.3 Connecting to Second Life

Let's describe the possible approaches to this problem. We have mostly 2 options, the first is to hack the second life client to make it communicate with our hardware, and the second is to make a piece of software, a sort of middleware, that interacts with both the second life client and our hardware.

Second life client hacked to make it communicate with our hardware

Figure 3.6: Second life client hacked to make it communicate with our hardware

Middleware that interacts with both second life client and our hardware

Figure 3.7: Middleware that interacts with both second life client and our hardware

3.4 Hacking the Second Life Client

The open source release of the Second Life viewer program by developer Linden Labs offers a rare opportunity to peer into the comparative strengths of closed and open source development models.Developers are free to modify and extend the SL client in order to add new features or fix existing bugs. Furthermore Linden Labs provide a test grid which developers can use to validate their modified clients before deploying on the main SL grid. Whilst this approach would allow developers to integrate hardware directly into SL a major impediment exists. Linden labs frequently distribute mandatory updates to their commercial SL client. Old versions of the SL client are prevented from connecting to the main grid until they have applied the necessary updates. While this is simply an inconvenience for users of unmodified clients, it represents a major impediment for those with modified clients as they must completely rebuild the client application to incorporate the latest updates and their original modification. However while modifying the client application to support hardware integration is in the short term impractical it does represent a more flexible solution longer term. Furthermore, Linden Labs encourages developers to submit modifications to the SL client for review and incorporation into subsequent releases of the commercial client.

Hacking the client basically consists in modifying the second life client code. It is written in C++ , so it is quite simple to modify its code adding the functions that realize the data exchange. It's quite easy (depending on the HW you want to connect to SL), because it converts our problem in a C# programming-solvable one.

The way of modifying the viewer is very easy because:
• Source code is available (the client was released as open source);
• Its easy to make changes to the client because the sources are available;
• You can examine everything on a system that is under your control.
As a result we have that almost all the examples we found make the connection with external applications modifying the Second Life Client.

Let's try to see positive and negative aspects of this approach:

Advantages:
"Easy coding": Second Life client is made in C++, so you can use all its potentialities and it's quite simple to exchange data with an external application.

Disadvantages:
"Code re-usability": Second Life client is frequently updated (at least once a week, but even more), so it's easily understandable why modifying it is not a "clean" way of programming.

How we can easily see, it could be a very good, simple, and fast way of solving the problem; but this solution is not opened to future development because a modification of the hardware to connect means re-writing the code from the beginning.

As an example we can see the project from the first link I showed before:

"
How does this work? All it takes is a few small code fragments in the right places

The Arduino code is trivial:

```
void setup() {
  Serial.begin(9600);
  pinMode(0,INPUT);
}
void loop() {
  delay(100);
  Serial.println(analogRead(0));
}
```

The in-world LSL code is trivial:

```
vector pos;
default
{
    state_entry()
    {
        llListen(42,"Matt Basiat",NULL_KEY,"");
        pos = llGetPos();
    }
    listen(integer channel, string name, key id, string message) {
        float val = (float)message;
        llSetPos(pos + <0,0,(val/100.0)>);
    }
}
```

Listing 3.1: Arduino code, script

The clever bit, such as it is, involves adding a bit of good old-fashioned Unix file-descriptor code to the SL client's idle loop (located in `viewer.cpp` in the source) that polls the Arduino. Once I've got the reading, all it takes to make my avatar speak to the server is the line:
gChatBar->sendChatFromViewer(reading, CHAT_TYPE_NORMAL, FALSE);
In total, I added about 150 lines of code to a single file to achieve this. Obviously this is just a concept demo, and there are plenty of places to take it from here, but I'm encouraged by the progress.
If you're curious about the details, here's the patch against `viewer.cpp` in the `newview` directory of the source. It's definitely not The Right Way To Do It - just the fastest route I could find into the heart of the code. It has hardcoded device names and other bad practice. I release it under the GPL, and must thank Tod Kurt for the use of his arduino-serial.c code and Massimo Banzi for everything he taught me about Arduino over the holiday season.
”

Here is the viewer.cpp implementation:

```
--- viewer.cpp.1	2007-01-12 14:11:05.000000000 +0000
+++ viewer.cpp	2007-01-11 16:43:27.000000000 +0000
@@ -541,6 +541,18 @@
 };
  */

+////
+// Arduino
+//
+
+#include <termios.h>
+#include <sys/ioctl.h>
+
+int gArduino = -1;
+int serialport_init();
+int serialport_write(int fd, const char* str);
+int serialport_read_until(int fd, char* buf, char until);
+
 //////////////////////
 //
 // Forward declarations
@@ -3966,6 +3978,24 @@
 			gAudiop->idle(max_audio_decode_time);
 		}

+		//////
+		// Arduino
+
+			if(gArduino == -1) {
+			    gArduino = serialport_init();
+				llinfos << "Inited arduino, fd: " << gArduino << llendl;
+				gChatBar->sendChatFromViewer("Arduino is online", CHAT_TYPE_NORMAL, FALSE);
+			} else {
+		char buf[256] = "/42 ";
+		int n = read(gArduino, buf+4, 100);
+                                                if(n > 0) {
+		    buf[n+4] = '\0';
+                                                        llinfos << buf << llendl;
+                                                        gChatBar->sendChatFromViewer(buf,               CHAT_TYPE_NORMAL,
FALSE);
+		}
+	      }
+		//llinfos << "Im in ur idle loop wasting yr cycles" << llendl;
+
 		// yield some time to the os if we are supposed to.
 		if(gYieldTime)
 		{
@@ -6519,3 +6549,101 @@
 }
 // JC - Please don't put code here.  Find the right file, perhaps
 // llviewermessage.cpp, and put it there.  Thanks!
+
+//////
+// Arduino
+
+int serialport_write(int fd, const char* str)
+{
+    int len = strlen(str);
+    int n = write(fd, str, len);
+    if( n!=len )
+        return -1;
```

```
+    return 0;
+}
+
+int serialport_read_until(int fd, char* buf, char until)
+{
+    char b[1];
+    int i=0;
+    do {
+        int n = read(fd, b, 1);  // read a char at a time
+        if( n==-1) return -1;    // couldn't read
+        if( n==0 ) {
+            usleep( 10 * 1000 ); // wait 10 msec try again
+            continue;
+        }
+        buf[i] = b[0]; i++;
+    } while( b[0] != until );
+
+    buf[i] = 0;  // null terminate the string
+    return 0;
+}
+
+// takes the string name of the serial port (e.g. "/dev/tty.usbserial","COM1")
+// and a baud rate (bps) and connects to that port at that speed and 8N1.
+// opens the port in fully raw mode so you can send binary data.
+// returns valid fd, or -1 on error
+int serialport_init()
+{
+    struct termios toptions;
+    int fd;
+
+    //fprintf(stderr,"init_serialport: opening port %s @ %d bps\n",
+    //        serialport,baud);
+
+    fd = open("/dev/tty.usbserial-A4000QpY", O_RDWR | O_NOCTTY | O_NDELAY);
+    if (fd == -1)  {
+        perror("init_serialport: Unable to open port ");
+        return -1;
+    }
+
+    if (tcgetattr(fd, &toptions) < 0) {
+        perror("init_serialport: Couldn't get term attributes");
+        return -1;
+    }
+            int baud = 9600;
+    speed_t brate = baud; // let you override switch below if needed
+    switch(baud) {
+    case 4800:   brate=B4800;   break;
+    case 9600:   brate=B9600;   break;
+// if you want these speeds, uncomment these and set #defines if Linux
+//#ifndef OSNAME_LINUX
+//    case 14400:  brate=B14400;  break;
+//#endif
+    case 19200:  brate=B19200;  break;
+//#ifndef OSNAME_LINUX
+//    case 28800:  brate=B28800;  break;
+//#endif
+    case 38400:  brate=B38400;  break;
+    case 57600:  brate=B57600;  break;
+    case 115200: brate=B115200; break;
+    }
+    cfsetispeed(&toptions, brate);
+    cfsetospeed(&toptions, brate);
+
```

```
+    // 8N1
+    toptions.c_cflag &= ~PARENB;
+    toptions.c_cflag &= ~CSTOPB;
+    toptions.c_cflag &= ~CSIZE;
+    toptions.c_cflag |= CS8;
+    // no flow control
+    toptions.c_cflag &= ~CRTSCTS;
+
+    toptions.c_cflag |= CREAD | CLOCAL;  // turn on READ & ignore ctrl lines
+    toptions.c_iflag &= ~(IXON | IXOFF | IXANY); // turn off s/w flow ctrl
+
+    toptions.c_lflag &= ~(ICANON | ECHO | ECHOE | ISIG); // make raw
+    toptions.c_oflag &= ~OPOST; // make raw
+
+    // see: http://unixwiz.net/techtips/termios-vmin-vtime.html
+    toptions.c_cc[VMIN]  = 0;
+    toptions.c_cc[VTIME] = 20;
+
+    if( tcsetattr(fd, TCSANOW, &toptions) < 0) {
+        perror("init_serialport: Couldn't set term attributes");
+        return -1;
+    }
+
+    return fd;
+}
  “
```

Listing 3.2: Arduino code, hacked viewer.cpp

3.5 Middleware

The second option is to design external software that runs parallel to the Second Life Client. This middleware has to manage the hardware, and to send/receive data from SL too. This is a more complicated way of solving the problem (than the "hacking" one), but it's a more flexible way. We want to build-up a flexible and long-lived architecture that permits us to manage new type of hardware simply adding the "driver" without re-writing the whole software. There isn't a "standard" way of doing it. For developing the middleware we have chosen Java as programming language because it allows to easily manage communication in both directions. Here you can see the top-level diagram of our software architecture:

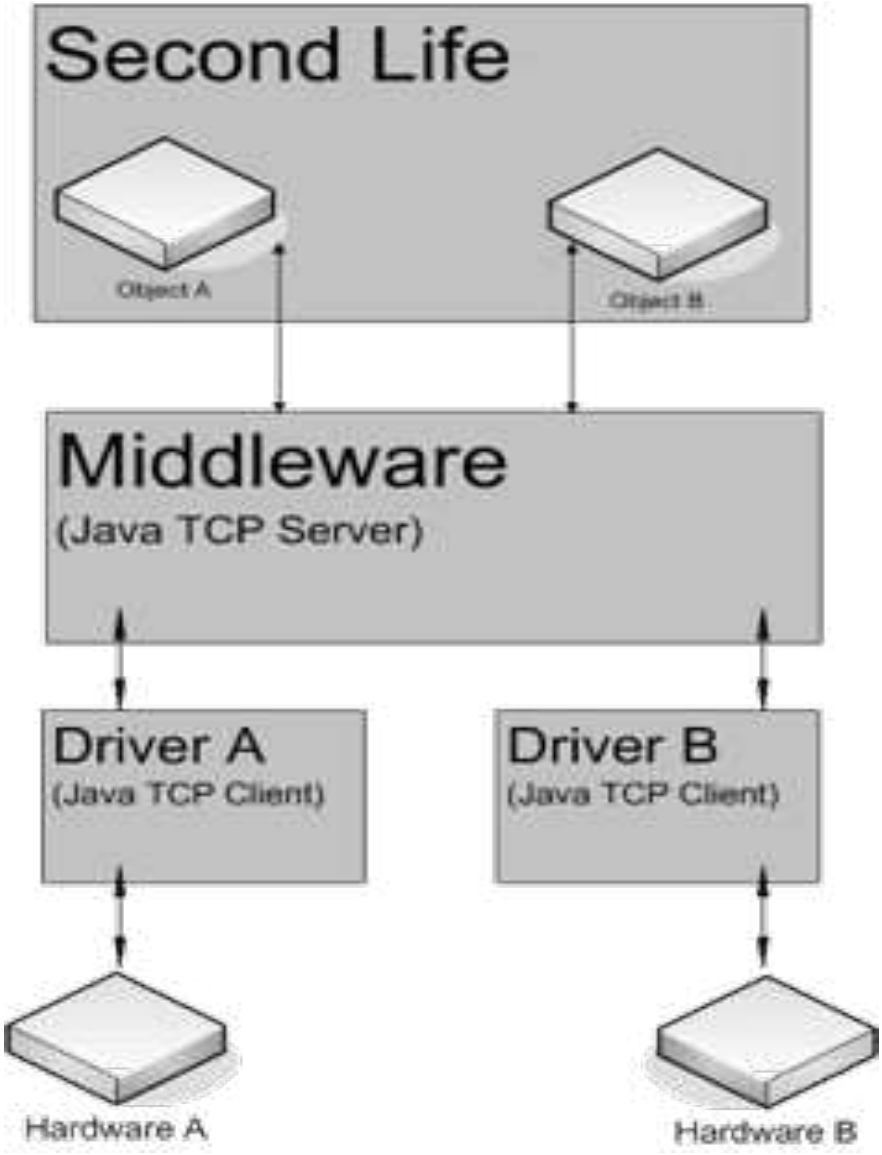


Figure 3.8: Top-level diagram of our software architecture

The Object A is the virtual equivalent of the Hardware A, the same is for the Object B with the Hardware B. The most important role in our architecture is the one of the “middleware”, it should be a piece of software that directly manages second life objects, and at the same time offers a list of “available functions” to the clients. This is the most important part of our architecture because it’s the one that never changes, it doesn’t matter if there are 3, 4, or 9 different hardware to connect to Second Life. The only thing that you have to do is writing a software that manages your hardware and connects via socket with the middleware, to make use of the functions that it offers. At the same time there should be an object in Second Life, to be connected to the real one. Basically, our idea is to build up “an application that permits to create a standard way to connect an hardware to Second Life, regardless of the kind of hardware”, doesn’t matter which is your hardware, the only thing that you have to do is to write a piece of (quite simple and in the major part always the same) lsl code, and the “driver”, as said before. At the beginning, maybe it means to make the connection between the hardware and Second Life a little bit more complicated than how it could be, but this is the price to pay to have a common standard available for all the hardware.

So the “middleware” is the real link between the real and the virtual world.
Just to explain it in a few words, the Second Life object lsl code has the goal to manage the SL object in the same way like the java driver has the goal to manage the real object (physical hardware) and both of them can make use of the function offered by the middleware.

As for the driver, it can simply create a java tcp connection with the middleware and send/receive specific commands to/from a particular sl object, this commands correspond to the functions that the middleware offers, and to actions that the Second Life object can do.
Looking at the SL side, the lsl script sends/receives the requests to/from the middleware, and in consequence modify the SL object status.

- # Advantage

Here are just 5 important characteristics of our software:

“Hardware Independent”

It is possible to connect different kind of hardware using the same Middleware. The architecture will be the same regardless of the kind of hardware.

“Multi-platform”

For implementing our middleware we can use Java language so our architecture can be made to work on multiple computer platforms whit different operating systems.
However, Java is limited in that it does not directly support system-specific functionality. JNI can be used to access system specific functions, but then the code is likely no longer portable. Java programs can run on at least the Microsoft Windows, Mac OS X, Linux, and Solaris operating systems, and so the language is limited to functionality that exists on all these systems. This includes things such as computer networking, Internet sockets, but not necessarily raw hardware input/output.

“Easy plug-in ”

If someone wants to connect a new hardware, he can do it simply making a new “driver” (java client that manages a specific hardware) and using the functions that our “java server” offers.

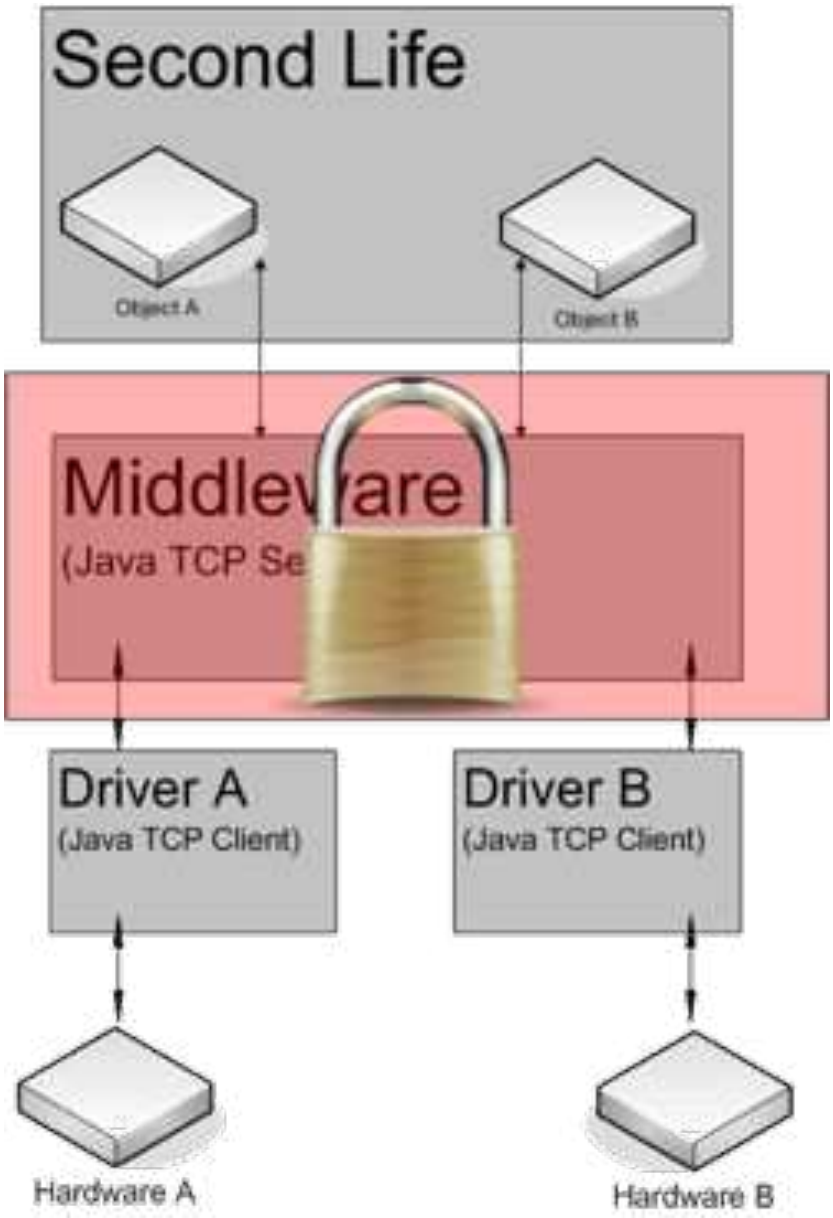


Figure 3.9: Advantage of our architecture, Easy plug-in

“Multi Client”

If we want to connect two or more clients that manages each one a different Second Life object we can do it and only one java server is running.

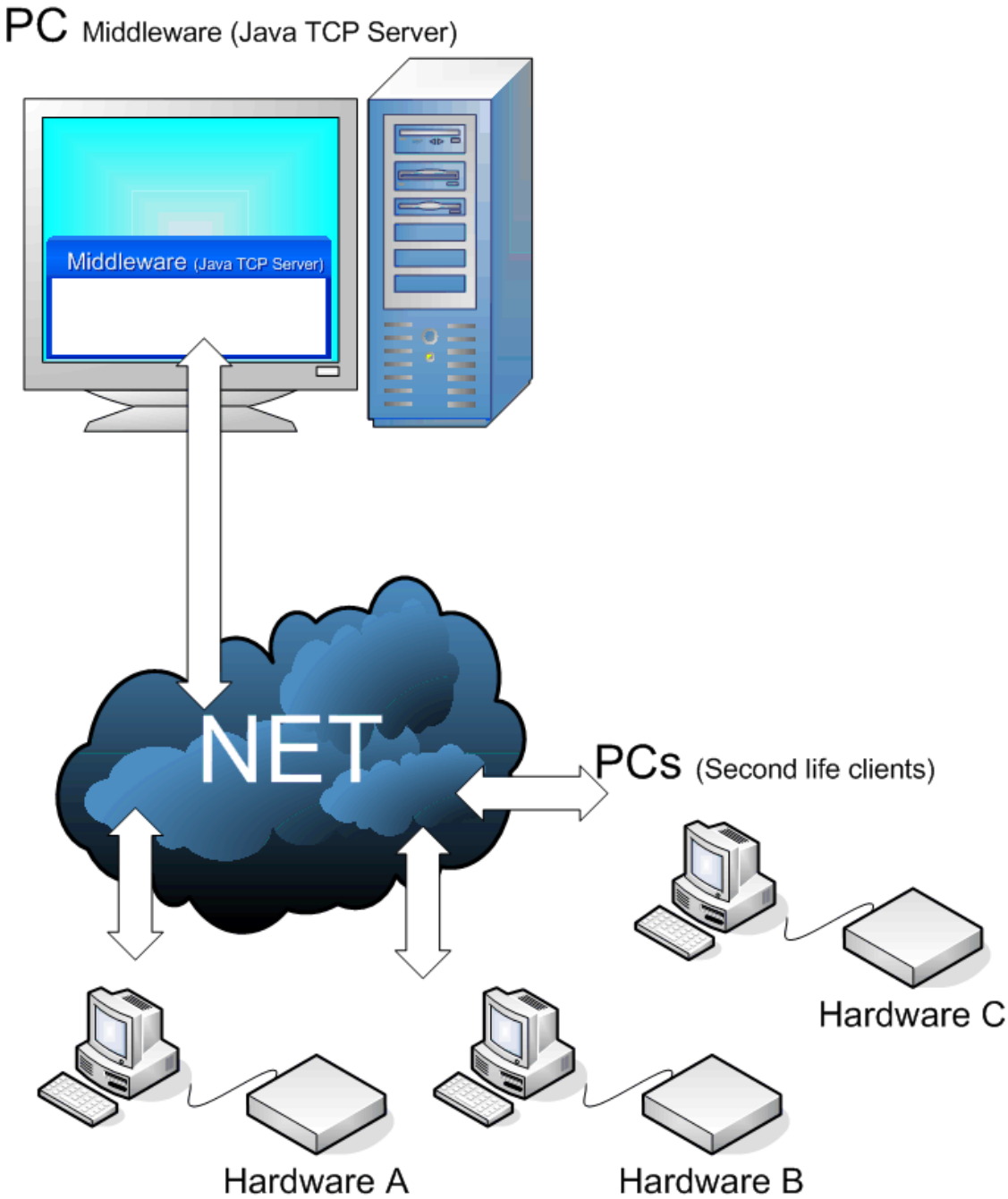


Figure 3.10: Advantage of our architecture, Multi Client

“Distributed System”
The “java server”, Second Life and each client (the “driver”) can run each one in a different machine, because they all communicate using the TCP/IP protocol.

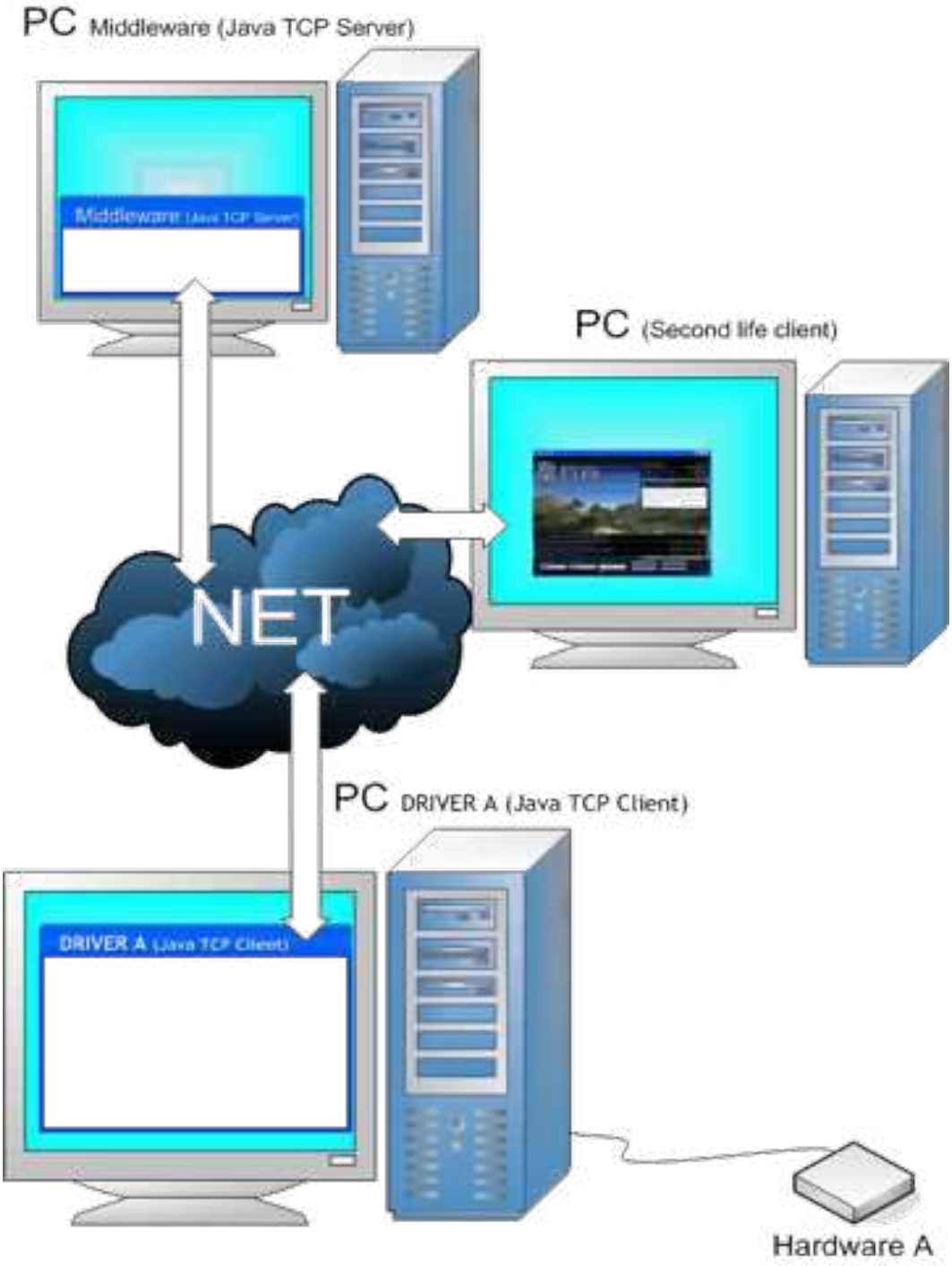


Figure 3.11: Advantage of our architecture, Distributed System

- **Disadvantage**

On the Second Life side, we have to write software using only the LSL Scripting code.

Fortunately, thanks to LSL scripting language offered by Second Life it is possible connect any external environment to the virtual world.

So, with proper scripting, objects can use the connectivity capabilities to exchange data with the outside world. At present, the scripting language provides three interfaces for communication with the Internet, e-mail, XML-RPC and HTTP requests. So, the only way to send/receive data to/from SL is using one of these three protocols:

Email :

E-mail can be sent both inside Second Life and to Internet addresses. To reduce load on the servers and prevent use for spam, there are limitations on how frequently an object can send e-mail and enforced delays for each piece of mail sent.
This obviously cannot be the way to communicate with an hardware, but it can be useful for some particular tasks (e.g. Passing an object-key to an external software)

HTTP Request :

It is one of the most useful functionality that LSL offers, it is fast and easy-to-use, but it is only a one-way communication. You can do http requests using the llHTTPRequest() function.
When a script calls to llHTTPRequest function, the language automatically associates a matching http_response event handling method which will get the resulting response.
An important aspect of http requests is that the headers contain complete specification of the object originating the request, including its owner, location and velocity. Thus any service addressed by the request has a lot of information suitable for enforcing security restrictions.

XML-RPC :

This is potential powerful functionality, and it was well implemented in Second Life only some months ago. It allows you to receive calls from external software but it is a one-way communication (you cannot make xml-rpc calls). Anyway it is the only way for external software to send data to SL.

As anyone can see, Second Life doesn't offer a standard, and easy to use, protocol to have any kind of communication/data exchange with any external application. The main reason is that at the beginning SL wasn't created (and consequentially designed) like an environment that needs data exchange with external applications; so, now, the main problem of anyone that wants to do this, is to set-up an application that makes use of these protocols.

Depending on the kind of the application to set-up, HTTPRequests can be a very useful and reliable solution to the data-exchange problem; because they are fast, easy to use, and overall you can make use of them in most programming languages(or platforms). Anyway, due to our goal, we have to send asynchronous data to and from SL (because no one can say exactly when the next event will happen!), so it's necessary to make use of the xml-rpc protocol.

3.6 Our choice

We have chosen not to modify the second life client, even if it is surely easier (and maybe faster). We want to build-up a flexible and long-lived environment that permits us to manage new type of hardware simply adding the “driver” without re-writing the whole software.

As seen before, someone has already tried to make this connection (Second Life-Hardware), but they used a different approach depending on the specific application. Instead, the architecture that I have designed is independent from the specific task.

Chapter 4
Demonstration: a possible application

4.1 Project overview

To explain how our architecture works we can imagine a possible application.
Using our architecture could be possible to realize applications for the home automation.
For example it is possible to solve the simplest "domestic" problem: turning on and off the houselights. Imagine that you're out of your home, if you want you can turn on the light of your bedroom, kitchen, or anything else directly from SL; so, practically, from anywhere in the world where you can find an internet connection! In the next paragraph I'll introduce the project and describe all the details useful to understand it.

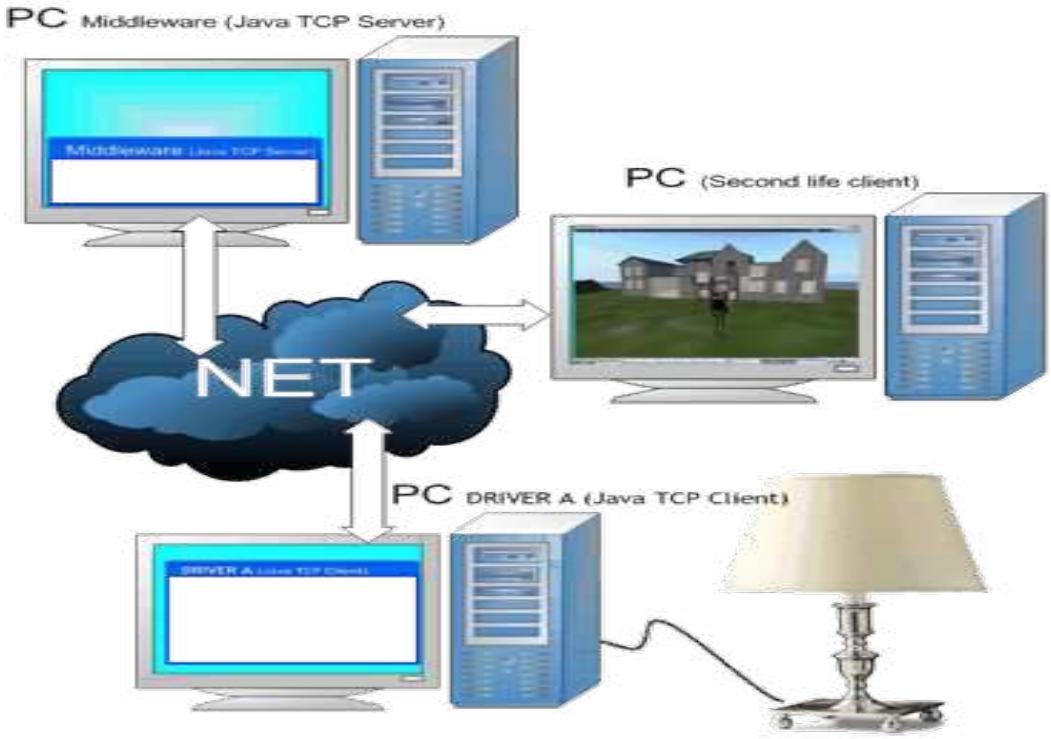


Figure 4.1: Demonstration: a possible application. A domotic problem, high level model

To avoid the power electronics problems, we can imagine that the light of every room is represented by a LED, so, for example, we can make use of the parallel port to manage a led (houselight); this means not only to be capable of turning them on/off if someone does it in Second Life, but also "to be sensible" to the "status modifications" caused by a real user in the real world.

Let's start analyzing the physical connection between the hardware and the pc, and with our software too.

4.2 Connecting the hardware to our architecture

For managing the LED we need we need 1 pin to write on it, and 1 pin to read from it (two ways control, as said before). So, analyzing the parallel port we can use pins 2 and 6 (D0 and D4), that are bidirectional:

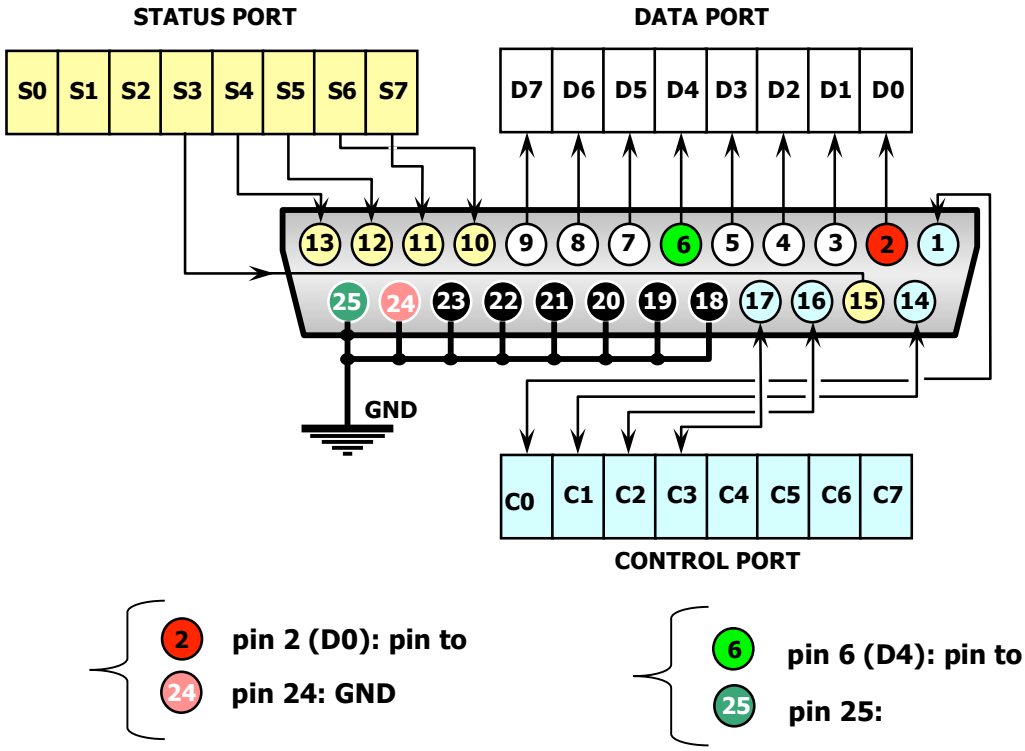
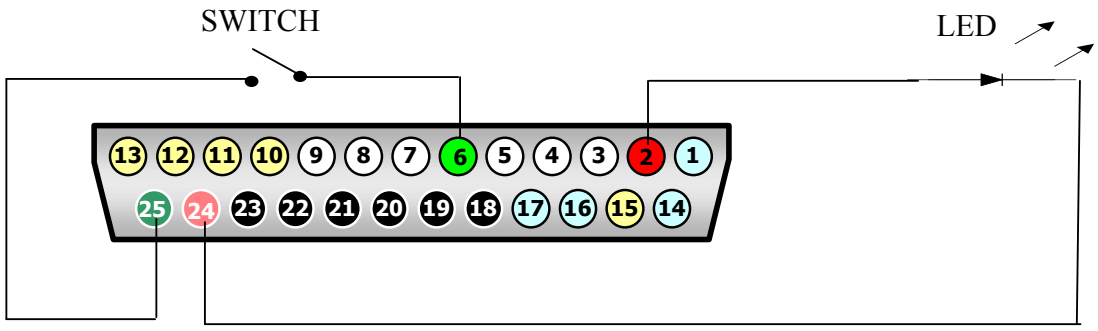


Figure 4.2: managing a LED via Parallel Port, electrical scheme

For our goal we can use a 3 volt LED so we do not need any resistor. Therefore the electrical scheme is very simple how we can see from the next figure:



An easy way to implement a “driver” that can control this circuit from below is as follows:

- assume the switch initially open and led off
- when the software starts to run it puts the pin 6 (D4) to an high logical value
-

In this case, the output of the parallel port will be this:
0X10 (the format is in Hexadecimal)
In practice the pin 6 (D4) is used as a control pin. The commutation of its logical value determines the turning on or the turning off of the led. For checking the logical state of the pin 6 (D04) we can schedule an ad hoc thread (thread to read).

Turning on
The led is off so the output of the parallel port is:
0X10
When the switch will be pressed the pin 6 (D4) will be short-circuited with the mass.
In this case, the output of the parallel port will be this:
0X00
Then the thread to read will write 0x11 on port and the LED turns on.

Turning off
The led is on so the output of the parallel port is:
0X11
When the switch will be pressed the pin 6 (D4) will be short-circuited with the mass.
In this case, the output of the parallel port will be this:
0X01
Then the thread to read will write 0x10 on port and the LED turns on.

4.3 Connecting hardware and driver

Every driver can communicate with its hardware in the way it wants. I have chosen to use JNI to manage the LED. The main reason is that it is connected to the pc via parallel port, and it's very easy to manage the parallel port in C#. So, we decided to use JNI to import in java some simple functions that permit to write to the parallel port.

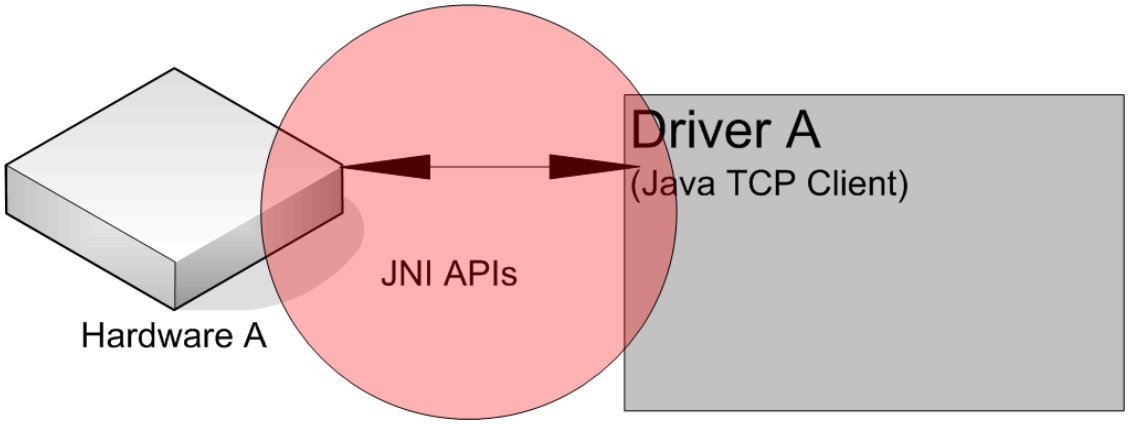


Figure 4.3: Connecting hw and driver

“

Java™ Native Interface (JNI) is a standard programming interface for writing Java native methods and embedding the Java™ virtual machine into native applications. The primary goal is binary compatibility of native method libraries across all Java virtual machine implementations on a given platform.

“

To use these API's under windows, you just have to copy a dll library in your Windows\System32 folder and then load that library to declare the functions that it contains as native ones into the java file, without implementing them. If you use Linux, java has many standard commands to manage the parallel port, and all the job is easier.

Truly I have used a modified version of Inpout32.DLL, to using JNI (Java Native Interface) for Windows 95/98/2000/XP. The new DLL is named "Jnpout32.DLL" to avoid confusion with the original.

Here there are 2 java classes that import some native functions and permit to easily read/write from the parallel port. In practice, the Driver does an import of these classes and uses the “output(Addr,datum)” and “input(Addr)” functions of the “pPort” class to manage the leds:

```
/* Definitions in the build of jnpout32.dll are:          */
/*   short _stdcall Inp32(short PortAddress);             */
/*   void _stdcall Out32(short PortAddress, short data);  */

public class ioPort
{
        // declare native methods of 'jnpout32.dll'
        // output a value to a specified port address
        public native void Out32(short PortAddress, short data);
        // input a value from a specified port address
        public native short Inp32(short PortAddress);

        // load 'jnpout32.dll' for package
        static { System.loadLibrary("jnpout32pkg");}
}
```

Listing 4.1: ioPort.java

```
package jnpout32;

// ** Derived from information provided on the web by Dr. Kenneth G. Schweller,
// ** ( http://web.bvu.edu/faculty/schweller/ ) and his Mars Rover project page.

public class pPort{
   ioPort pp;            // wrapper class for 'Jnpout32.dll'
                         // with methods:
                         //    int Out32(int port, int value);
                         //    int Inp32(int port);
   short portAddress;          // address of data port
   short currentVal;           // current value of port bits

   public pPort()
   {
            pp = new ioPort();
            portAddress = (short)0x378;     // Hex Address of Data Byte of PC Parallel Port
            setAllDataBits((short)0);       // initialize port bits to 0
            currentVal = 0x00;
   }

   // wrap ParallelPort output method
   public void output(short port, short value) { pp.Out32(port, value); }

   // wrap ParallelPort input method
   public short input(short port) {  return pp.Inp32(port);  }

    // output to default Data port
    public void output(short value) { pp.Out32(portAddress, value); }

    // input from default Data port
public short input(){ return pp.Inp32(portAddress); }
```

Listing 4.2: pPort.java

4.4 Connecting Driver and Middleware

For connecting the drivers (Java TCP Clients) to our middleware (Java TCP Server) we have to use:
- standard java APIs for client-server applications.
- sockets and streams to pass data.
Notice that the server can manage Connection/Disconnection of many drivers(clients).

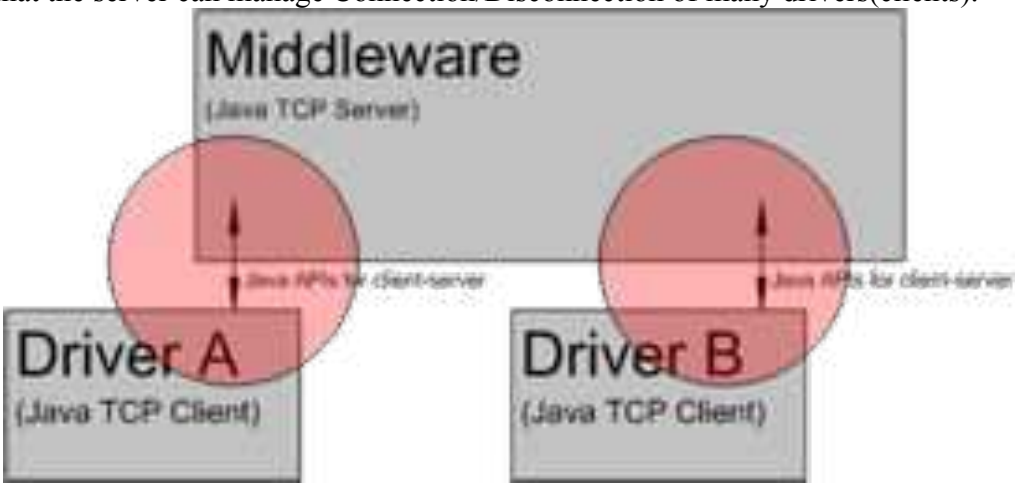


Figure 4.4: Connecting Driver and Middleware

Here is an example of Server that, when called by the client, reads a line and send it back to the client:

```
ssocket=new ServerSocket(port);
socket=ssocket.accept();
toClient=new DataOutputStream(socket.getOutputStream());
fromClient=new BufferedReader(new InputStreamReader(socket.getInputStream()));
read=fromClient.readLine();
toClient.writeBytes(read + "\n");
fromClient.close();
toClient.close();
socket.close();
```

4.5 Connecting Middleware and Virtual World

As seen before, we are obliged to use HTTPRequests for outgoing requests (seen from the SL side) and XML-RPC for incoming requests. So, at the beginning the scheme we wanted to use was this:

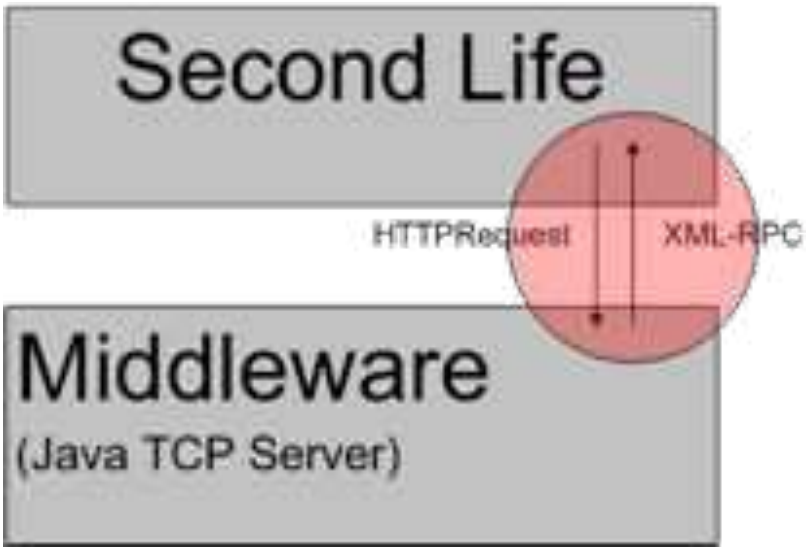
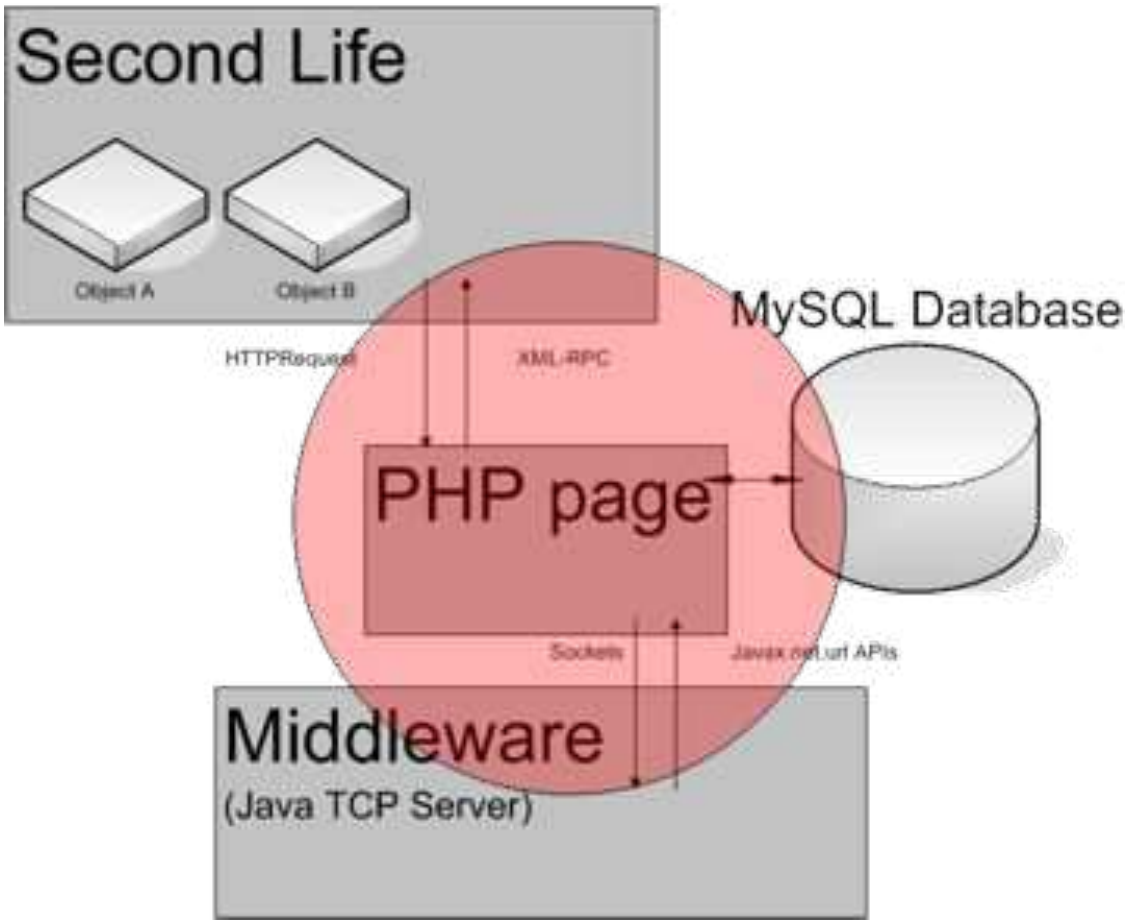


Figure 4.5: Connecting Middleware and Virtual World

To accept Http Requests from a java application, we need a Servlet/Jsp page, (Tomcat), but this adds a layer between the middleware itself and SL. Moreover we have to add another problem to this one, in that the XML-RPC libraries gave us many stability problems.
As a solution we decided to add a "php layer" between the middleware and second life. Accepting Http Requests to php pages is immediate, and php can easily send data to the java server. In addition, we can make use of the php pages to send xml-rpc requests from them instead of java, it's easier and we have to write just a few lines of code.



This (adding php between the middleware and SL) doesn't make the system slow (the bottleneck is the xml-rpc implementation in SL) and allows us to use a database, that could be useful for various tasks, like for example creating a logfile with all the recent events, or anything else.

4.6 Architecture and protocol definitions

So, the definitive scheme of our architecture (with protocol definitions) is:

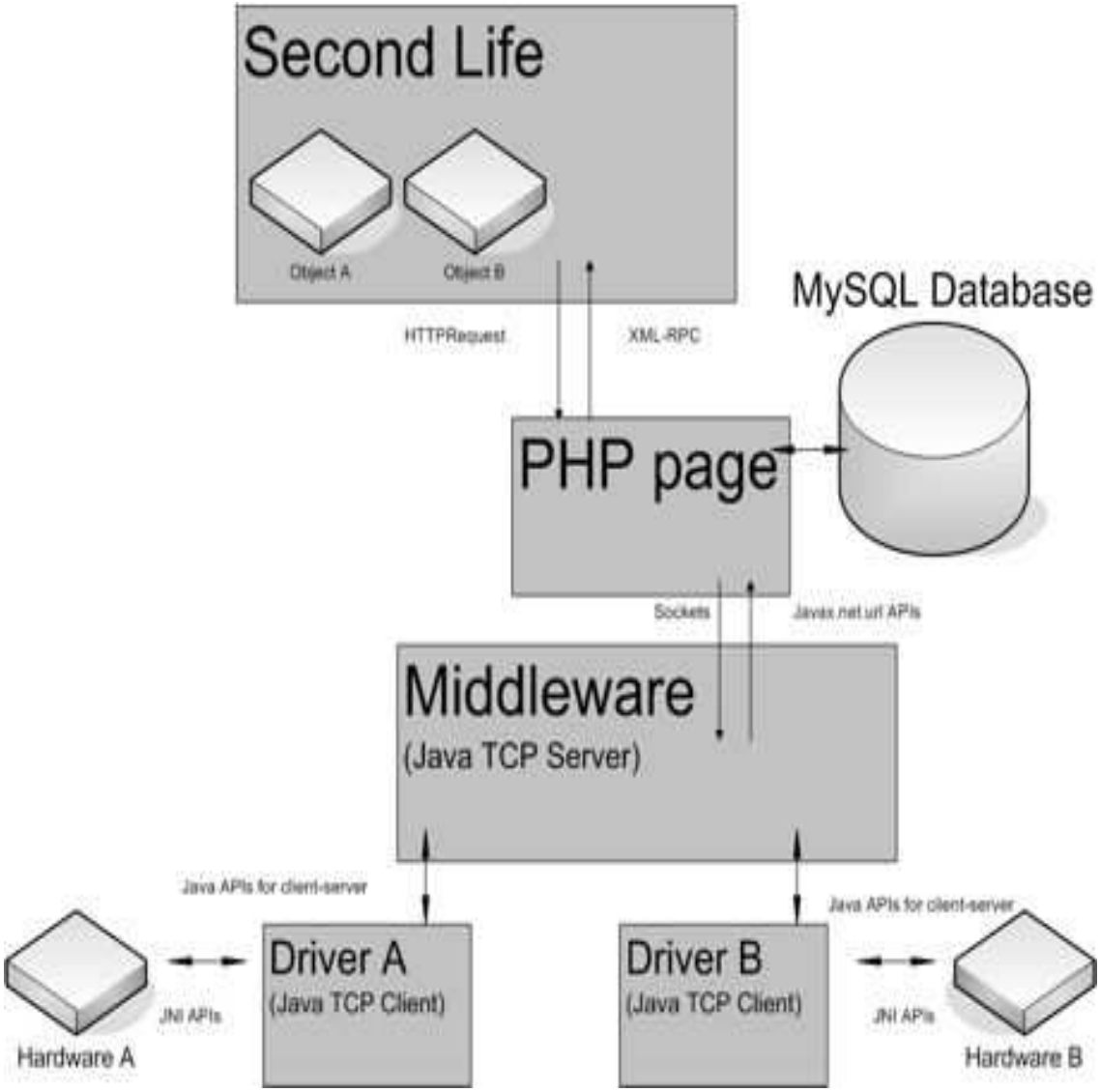


Figure 4.6: Architecture and protocol definitions

As said before, the "driver" has to communicate both with the server (to make use of the functions that it offers) and with the hardware (to manage it). It's the only piece of software that someone has to write if he wants to connect their hardware to SL, the rest of the architecture is always the same. Each piece of hardware has its own driver, and they all use the same middleware.

4.7 Class diagram

For understanding how the system is structured a class diagram is very useful. In the Unified Modelling Language (UML), a class diagram is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, and the relationships between the classes.

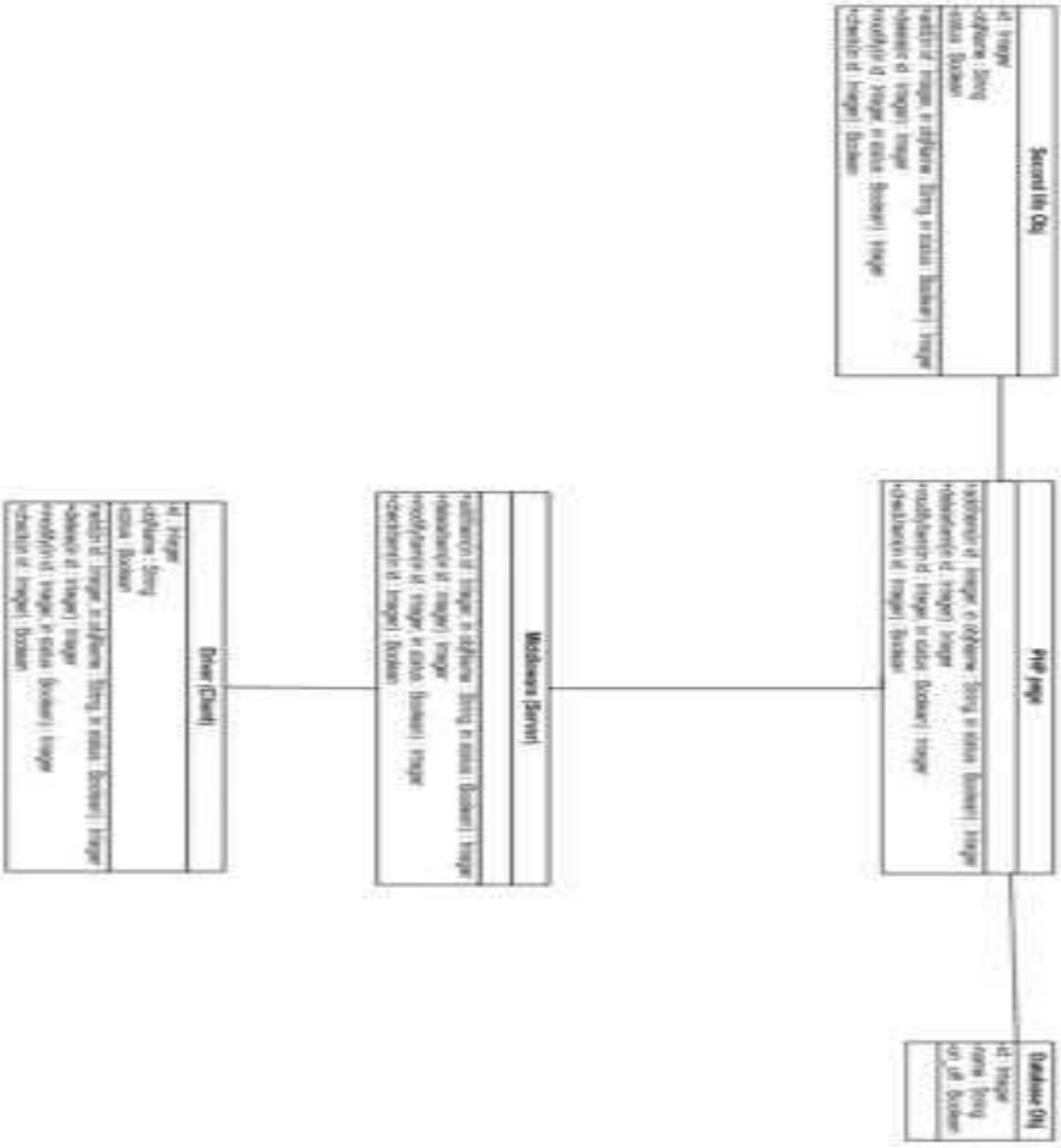


Figure 4.7: Class diagram

This diagram is not a proper class diagram, but it helps understanding the entities of the architecture, and the links between them, better, how they can "communicate" between themselves.

As you can see, every entity (the lsl script, the php page, the middleware, and the driver) has the standard 4 functions/actions (add/modify/delete/check). Every function correspond to a php page that can modifies the database, and communicate with both the middleware(via sockets) and the lsl script(via xml-rpc).

So, if the event took place in the real world, the driver will call the proper function (connection/disconnection, new hardware, status changed) in the middleware, that will call the proper php page(its code will send an xml-rpc to the sl object, identified by the obj_key). The sl object receives the information and changes its status. The same thing happens in the other way: if something happens to the virtual object, it will call a particular php page(depending on which is the event), that will update the database, and communicate the status modification to the driver(via middleware-sockets).

4.8 "Handshake phase"

To explain what really happens in the, it's very useful to know that, how we already said, every item in the Second Life universe is referred to as an asset. This includes the shapes of the 3D objects known as primitives, the digital images referred to as textures that decorate primitives, digitized audio clips, avatar shape and appearance, avatar skin textures, LSL scripts, information written on notecards, and so on. Each asset is referenced with a universally unique identifier or UUID.
For this reason is necessary to have a sort of "handshake" before that the system start to work normally. So, I will describe the "handshake" phase, and then, once the system is fully working, can analyze how it works.

Chronological sequence of events:
- The Driver is started, when the user clicks "connect" it send a "new hardware" request to the middleware, with the real object name(decided by the user). Then it waits for the middleware ack.
- The middleware receives the request and checks if there already is an object with that name. If so, it notifies this to the driver, that sends an error message to the user("The object already exists, choose another name"). A new request(with a new name) by the driver is needed. If the middleware doesn't find that name in the database, it creates the object in the database and puts name and status into the proper table (the id is given automatically by the database).
- OBJ_KEY field is empty[OBJ_KEY is the unique identifier for a sl object, given by SL itself, and is the only way to identify the object – so, it's strictly necessary to do xml-rpc's].
- SL object checks if there is already an item in the database with its name and only if it exists, it reads the table id and stores the Obj_Key. Then it reads the status of the real object and modifies consequentially its own one.
- When SL adds the Obj_key, the php page communicates to the middleware(via sockets) that the obj_key for that particular object is stored, and sends it to the middleware.
- The middleware provides to send the OBJ_KEY to the proper Driver, together with the ack.

- The Driver receives the ack. Now the driver is ready to communicate to the sl object.
Seen from the Driver-side:

1. Hardware plugs-in
2. Java client starts
3. Java Server recognizes the client and adds a new item in the database (NAME and STATUS are given by the client, ID is given by the database, OBJECT_KEY is empty)
4. A new object in SL is created
5. Java Client waits for its Second Life Object to connect and add its OBJECT_KEY in the database.
6. When OBJECT_KEY is added the php page will call the middleware and it will notify it to the Client.
7. The Client will store it and normally work.

Seen from the Second Life side:
1. When the object is created it will check if the database contain an item with its name.
2. If not, it will stop and check another time when someone will touch it.
3. If yes it will take the ID and the STATUS, and it will set the OBJECT_KEY.
5. Now it normally works.

4.9 From Second Life to Real world and vice-versa

Let us explain the connection from Second Life to Real World.
Suppose we have the virtual equivalent of an LED in Second Life. For our demo we can create a simple sphere that changes its color when the real LED is turned off/on.
When someone touches the virtual LED in Second Life, the object makes an http request to know if there already is an object with its name in the database, if not the object prints an error message on chat channel ("The real object is not connected, please connect it first and try again") and waits until another avatar touches it. Instead, if there already is an object with its name in the database, it takes its id, stores it as a local variable and updates its status with the one of the real LED's (stored in the database). Then, if someone touches the object another time, it immediately changes its status (turns on/off) and make an http request to a particular php page, giving as a parameter the object id and the new status.

Now we will explain the back connection (what happens if someone turns the led on by the switch). Before turning the led on or off, it's needed to have the server running. So, if the server is not started, it will be started, then it's the time to run the client.
The first thing that a client does is to check if there is another piece of hardware with his name, if so it returns an error message, if not it will call a php page to add a new item in the database with its name and status. After this you can turn the led on/off, eg. if you for example turn it on, it will send the corresponding request to the middleware.
The middleware will call the proper php page with the id of the object and the new status, so the php page can update the object's status on the table of the online hardware, and after this it will send an xml-rpc request to SL. The SL object will receive the xml-rpc request and properly update its status. For better understanding the sequence of events it is very useful to have a sequence diagram:

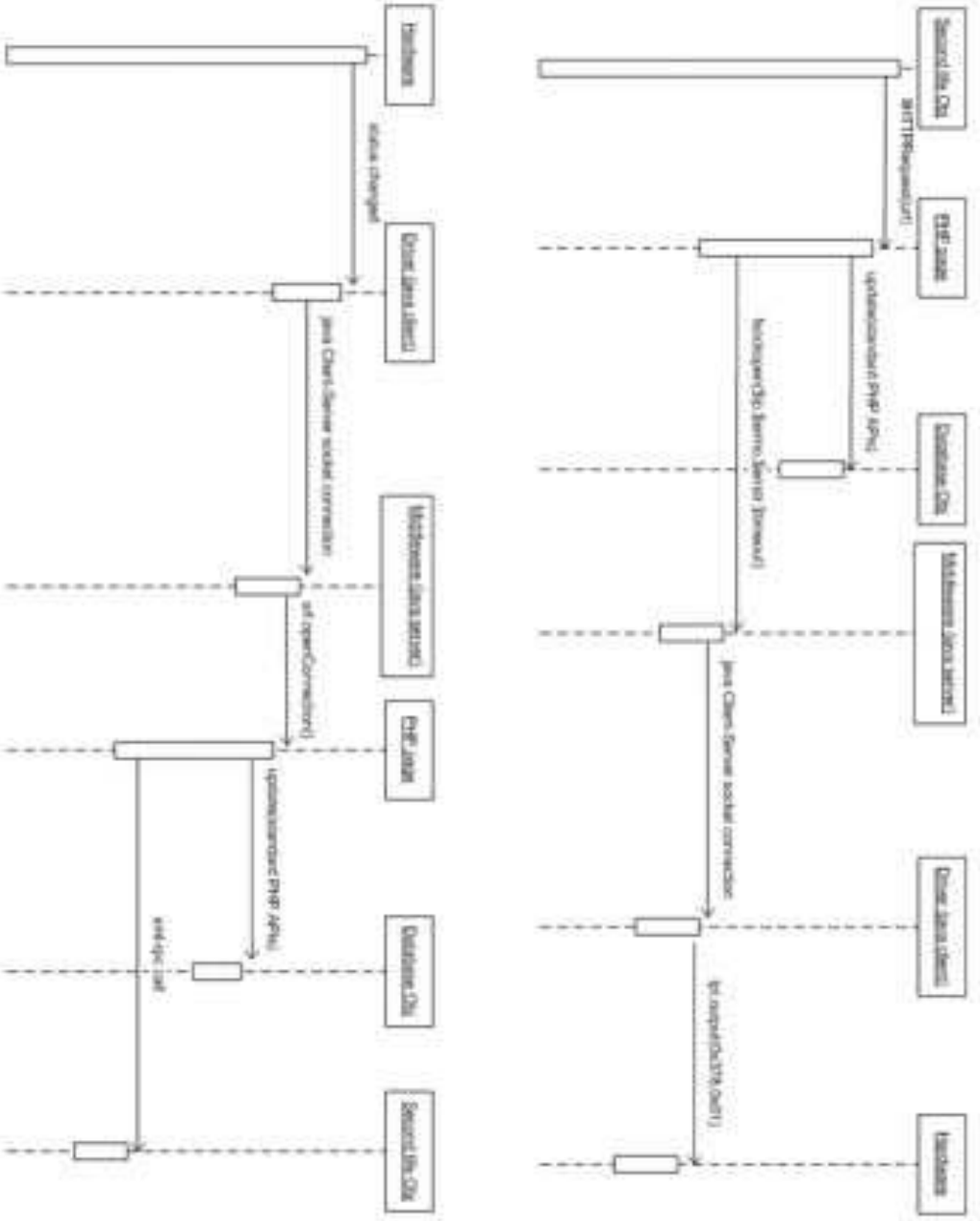


Figure 4.8: Sequence diagram

4.10 Code

Here is the lsl script inside a single virtual led in Second Life. It allows the object to change its status if receives a proper xml-rpc request, and at the same time when touched makes an http request to a php page:

```
string domain=      "http://hardware.secondlife-educator.com/sl/";
string add=         "http://hardware.secondlife-educator.com/sl/SLaddItem.php?";
string del=         "http://hardware.secondlife-educator.com/sl/SLdeleteItem.php?";
string mod=         "http://hardware.secondlife-educator.com/sl/SLmodifyItem.php?from=sl&";
string chk_name=    "http://hardware.secondlife-educator.com/sl/SLcheckItemByName.php?";
string chk_id=      "http://hardware.secondlife-educator.com/sl/SLcheckItemById.php?";
string add_obj_ref= "http://hardware.secondlife-educator.com/sl/SLaddObjRef.php?objRef=";

string objName;
key req_id;
key req_status;
key req_obj_ref;
key RemoteChannel; // my llRemoteData channel
integer status;
integer id;


default
{
   state_entry()
   {
      llOpenRemoteDataChannel();
      objName= llGetObjectName();
      llSay(0, "Hello, i'm "+objName+"!");
      req_id= llHTTPRequest(chk_name+"name="+objName,[],"");
   }

   touch_start(integer total_number)
   {
      req_id= llHTTPRequest(chk_name+"name="+objName,[],"");
   }

   remote_data(integer type, key channel, key message_id, string sender, integer ival, string sval)
   {
      if (type == REMOTE_DATA_CHANNEL)
      { // channel created
         RemoteChannel = channel;
         llSay(0, "Ready to receive XML-RPC requests");
      } else llSay(0,"Unexpected event type"+ (string) type + (string) channel + (string) message_id + sender + (string)
ival + sval);
   }


   http_response(key request_id, integer status, list metadata, string body)
   {
      if (request_id == req_id)
      {
         if(body == "-1")
         {
```

```
            llSay(0,"The real object is not connected, please connect it first and try again");
        }
        else
        {
            id = (integer) body;

             req_obj_ref= llHTTPRequest(add_obj_ref+(string) RemoteChannel+"&id="+(string) id ,[],"");
        }
    }
  else if(request_id == req_obj_ref)
  {
    if(body == "-1")
          {
            llSay(0,"An error occurred while trying to add the object reference to the db, please check the db and try
again");
          }
          else
          {
            req_status= llHTTPRequest(chk_id+"id="+(string) id,[],"");
          }
  }
    else if(request_id == req_status)
    {
        status= (integer) body;
        if(status == '1')
        {
            llSay(0, "Turning on");
            state on;
        }
        else if(status == '0')
        {
            llSay(0, "Turning off");
            state off;
        }
        else
            llSay(0,"Sorry a database error occurred, please check it and try again");
    }
    else
        llSay(0,"error");
  }

}



state on
{
  state_entry()
  {
    llSetColor(<1.0,1.0,1.0>,ALL_SIDES);
  }

  remote_data(integer type, key channel, key message_id, string sender, integer ival, string sval)
  {
    if (type == REMOTE_DATA_REQUEST)
    { // handle requests sent to us
```

```
            // Set the default reply to the received data
            string stringPortionOfReply = "Default Reply";


            if(ival== 000)
            {
                llSay(0,"Real object is disconnected, restarting...");
                state default;
            }

            if(ival == 0)
            {
                status = 0;
                integer intPortionOfReply = 0;
                llRemoteDataReply(channel, message_id, stringPortionOfReply, intPortionOfReply);
                llSay(0,"Turning off");
                state off;
            }

        } else llSay(0,"Unexpected event type"+ (string) type + (string) channel + (string) message_id + sender + (string)
ival + sval);
    }

    touch_start(integer total_number)
    {llHTTPRequest("http://hardware.secondlife-educator.com/sl/SLmodifyItem.php?from=sl&id="+(string)
id+"&on_off=0",[],"");
        llSay(0, "Turning off");


        state off;
    }
}



state off
{
    state_entry()
    {
        llSetColor(<0.0,0.0,0.0>,ALL_SIDES);
    }

    remote_data(integer type, key channel, key message_id, string sender, integer ival, string sval)
    {
        if (type == REMOTE_DATA_REQUEST)
        { // handle requests sent to us
            // Set the default reply to the received data
            string stringPortionOfReply = "Default Reply";


            if(ival== 000)
            {
                llSay(0,"Real object is disconnected, restarting...");
                state default;
            }
```

```
        if(ival == 1)
        {
            status = 1;
            integer intPortionOfReply = 1;
            llRemoteDataReply(channel, message_id, stringPortionOfReply, intPortionOfReply);
            llSay(0,"Turning on");
            state on;
        }

    } else llSay(0,"Unexpected event type"+ (string) type + (string) channel + (string) message_id + sender + (string)
ival + sval);
  }

  touch_start(integer total_number)
  {
    llSay(0, "Turning on");
    llHTTPRequest("http://hardware.secondlife-educator.com/sl/SLmodifyItem.php?from=sl&id="+(string)
id+"&on_off=1",[],"");
    state on;
  }
}
```

Listing 4.3: Lsl script inside a single virtual led in Second Life

Here is the code ragrding the php page that, when called, will update the status of the corresponding object into the mysql database, and immediately after this it will open a socket connection to the middleware, and send the id of the object and the new status to it, after this it will close the connection. Php page code example:

```
<?PHP
    session_start();
    include ("manageUserData.php");   //it is just for managing the access to the internet site
    include ("IXR_Library.inc.php");
    include ("javaData.php");                    //it contain informations and the IP address of the machine were
                                                 the middleware is running
    $ch=$_GET['id'];
    @$new_on=$_GET['on_off'];
    @$new_name=$_GET['name'];
    @$objRef=$_GET['objRef'];
    $db=connect();
    if($new_name==""){
            if(isset($_GET['on_off'])){
                    $query= "UPDATE status SET on_off='$new_on' WHERE id='$ch' ";
                    $res = mysql_query($query,$db) or die("Errore nella query:".mysql_error());
            }
    }
    else{
            if(isset($_GET['on_off'])){
                    $query= "UPDATE status SET name='$new_name',on_off='$new_on' WHERE id='$ch' ";
                    $res = mysql_query($query,$db) or die("Errore nella query:".mysql_error());
            }
            else{
                    $query= "UPDATE status SET name='$new_name' WHERE id='$ch' ";
                    $res = mysql_query($query,$db) or die("Errore nella query:".mysql_error());
            }
    }
```

```
	disconnect($db);
	echo $ch;

	@$from = $_GET['from'];
	if($from == 'java'){
		$data = '<?xml version="1.0"?>';
		$data .= '<methodCall>';
		$data .= '<methodName>llRemoteData</methodName>';
		$data .= '<params><param><value><struct>';
		$data .= '<member><name>Channel</name><value><string>'.$objRef.'</string></value></member>';
		$data .= '<member><name>IntValue</name><value><int>'.$new_on.'</int></value></member>';
		$data .= '<member><name>StringValue</name><value><string>Paolino Paperino</string></value></member>';
		$data .= '</struct></value></param></params></methodCall>';

		$fp = fsockopen("xmlrpc.secondlife.com", 80);
		fputs($fp, "POST /cgi-bin/xmlrpc.cgi HTTP/1.1\r\n");
		fputs($fp, "Host: xmlrpc.secondlife.com\r\n");
		fputs($fp, "Content-type: application/x-www-form-urlencoded\r\n");
		fputs($fp, "Content-length: ". strlen($data) ."\r\n");
		fputs($fp, "Connection: close\r\n\r\n");
		fputs($fp, $data);
		while(!feof($fp)) {
			$res .= fgets($fp, 128);
		}
		fclose($fp);
	}

	if($from == 'sl'){
		$fp = fsockopen($ip, $port, $errno, $errstr, 30);
		fwrite($fp,"?PHP?\n");
		fwrite($fp,"??OBJMOD??\n");
		fwrite($fp, (string) $ch);
		fwrite($fp, "\n");
		fwrite($fp, (string)$new_on);
		}
	fclose($fp);

?>
```

Listing 4.4: php page that updats the status of the corresponding object

The middleware is a java tcp server waiting for socket connections on a particular port, every time it receives a connection request, it launches a new thread that will manage it. This thread will receive the request (made by an id and a string that represents the command), and in our case (the SL object status is changed) it will send to the corresponding “driver” (that is a java client) the new status. It will wait for new requests, unless the request is a disconnection one.

```
public ClientThread(Socket s,String nameH,InetAddress ipH,int portH)
  {
    this.s=s;
    this.nameH=nameH;
    this.ipH=ipH;
    this.portH=portH;
    try
    {
```

```
      toClient=new DataOutputStream(socket.getOutputStream());
      fromClient=new BufferedReader(new InputStreamReader(socket.getInputStream()));
    }
    catch(Exception e){}
  }

  public void run()
  {
    try
    {
      while(true)
      {
        read=fromClient.readLine();
        if(read==null)
          break;
                        if(read.equals("?1?"))
                        {
                                Status=1;
                                display.append("\nRequest 'switch ON' received from: \n");
                                display.append("Host name: "+nameH+"\tIp address: "+ipH+"\tRemote port:
"+portH+"\n\n");
                                url = new URL (mod+"on_off="+Status+"&id="+id+"&objRef="+objRef);
                                URLConnection url_conn = url.openConnection();
                                ins = (InputStream) url_conn.getContent();
                                isr = new InputStreamReader(ins);
                                rdr = new BufferedReader (isr);
                                rdr.readLine();
                        }

                        if(read.equals("?0?"))
                        {
                                Status=0;
                                display.append("\nRequest 'switch OFF' received from: \n");
                                display.append("Host name: "+nameH+"\tIp address: "+ipH+"\tRemote port:
"+portH+"\n\n");
                                url = new URL (mod+"on_off="+Status+"&id="+id+"&objRef="+objRef);
                                URLConnection url_conn = url.openConnection();
                                ins = (InputStream) url_conn.getContent();
                                isr = new InputStreamReader(ins);
                                rdr = new BufferedReader (isr);
                                rdr.readLine();

                        }

                        if(read.equals("?NH?"))
                        {
                                name= fromClient.readLine();
                                url = new URL (chk_name+"name="+name);
                                URLConnection url_conn = url.openConnection();
                                ins = (InputStream) url_conn.getContent();
                                isr = new InputStreamReader(ins);
                                rdr = new BufferedReader (isr);
                                if((cod_html= rdr.readLine())!=null)
                                {
                                        if(!cod_html.equals("-1"))
                                        {
```

```
                                        display.append("\nERROR: there is already an item name '"+name+"' in the database,try again\n");
                                        toClient.writeBytes("??EXISTING_NAME??\n");
                                }
                                else
                                {
                                        toClient.writeBytes("??CREATINGOBJ??\n");
                                        Status= Integer.parseInt(fromClient.readLine());
                                        url = new URL (add+"name="+name+"&on_off="+Status);
                                        url_conn = url.openConnection();
                                        ins = (InputStream) url_conn.getContent();
                                        isr = new InputStreamReader(ins);
                                        rdr = new BufferedReader (isr);
                                        id = Integer.parseInt(rdr.readLine());
                                        display.append("\nSuccessfully created in the database an item with name: '"+name+"' and id: '"+id+"'\n");
                                }
                        }
                }

                if(read.equals("?PHP?"))
                {
                        display.append("\nphp request\n");
                        ClientThread aux;
                        String clientName;
                        int clientId;
                        String oRef;
                        DataOutputStream toC;
                        BufferedReader fromC;
                        php = true;
                        name = new String("##php_thread##");
                        String requestType = fromClient.readLine();
                        if(requestType.equals("??OBJREF??")){
                                display.append("\nOBJ Ref request\n");
                                id = Integer.parseInt(fromClient.readLine());
                                objRef = fromClient.readLine();
                                display.append("\n"+id+"\n"+objRef+"\n");
                                for(int i=0; i<Clients.size(); i++)
                                {
                                        aux = Clients.elementAt(i);
                                        clientId = aux.id;
                                        clientName = aux.name;
                                        toC = aux.toClient;
                                        fromC = aux.fromClient;
                                        if(clientId==id && !name.equals(clientName))
                                        {
                                                aux.objRef = this.objRef;
                                                toC.writeBytes("??OK??\n");
                                        }
                                }
                        }
                        else if(requestType.equals("??OBJMOD??")){
                                id = Integer.parseInt(fromClient.readLine());
                                Status = Integer.parseInt(fromClient.readLine());
                                for(int i=0; i<Clients.size(); i++)
                                {
```

```
                                        aux = Clients.elementAt(i);
                                        clientId = aux.id;
                                        clientName = aux.name;
                                        toC = aux.toClient;
                                        fromC = aux.fromClient;
                                        if(clientId==id && !name.equals(clientName))
                                        {
                                                toC.writeBytes("?PHP?\n");
                                                toC.writeBytes(Status+"\n");
                                                aux.Status= this.Status;
                                        }
                                }
                        }
                        else
                                display.append("\nError, unknown request received from php\n");
                }
        }

                if(php)
                {
                        display.append("\nDisconnection request received from the Php Server \n");
                }
                else
                {
                        display.append("\nDisconnection request from: \n");
            display.append("Host name: "+nameH+"\tIp address: "+ipH+"\tRemote port: "+portH+"\n\n");
                        url = new URL (del+"choice="+id);
                        URLConnection url_conn = url.openConnection();
                        ins = (InputStream) url_conn.getContent();
                        isr = new InputStreamReader(ins);
                        rdr = new BufferedReader (isr);
                        rdr.readLine();
                }
            fromClient.close();
            toClient.close();
            socket.close();
    }
    catch(Exception e){display.append(e+"\n");}
  }
}
```

Listing 4.5: middleware.java

The Driver is a java tcp client that manages a particular hardware. When it receives the command from the server, it modifies the led status (i.e. Turns it on/off). Here is the code inside the client(Driver) that receives the request by the server to modify its status:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.net.*;
import java.io.*;
import java.util.*;
import jnpout32.*;

public class Client
{
```

```
static String objName= new String("Prova");
static short datum;
static short read_datum;
static short Addr=0x378;
static pPort lpt;
static myFrame frame;
static Container c;
static JPanel north;
static JPanel center;
static JPanel center_north;
static JPanel south;
static JLabel label;
static JTextArea display;
static JTextField connection;
static JTextField showStatus;
static JButton connect;
static JButton disconnect;
static JButton Switch;
static JButton exit;
static String nameHost=new String("localhost");
static int port=2121;
static Socket socket;
static DataOutputStream toServer;
static BufferedReader fromServer;
static String aux=new String("");
static boolean status=false;
static boolean interruttore=false;
static readFromServer rfs;
static readFromPort rfp;
static boolean ck=true;

public static synchronized void write_datum(short temp)  throws Exception
{
  datum=temp;
}
static void do_read()
{
  // Read from the port
  read_datum = (short) lpt.input(Addr);
  // Notify the console
  System.out.println("Read Port: " + Integer.toHexString(Addr) +" = " +  Integer.toHexString(read_datum));
}

static void do_write()
{
          // Notify the console
  System.out.println("Write to Port: " + Integer.toHexString(Addr) +" with data = " +  Integer.toHexString(datum));
  //Write to the port
  lpt.output(Addr,datum);
}

static class myFrame extends JFrame implements ActionListener
{
  public myFrame(String name)
  {
    super(name);
```

```
  setDefaultCloseOperation(EXIT_ON_CLOSE);
  Toolkit tk=Toolkit.getDefaultToolkit();
  Dimension screenSize=tk.getScreenSize();
  setBounds(screenSize.width/3,screenSize.height/3,screenSize.width/2,screenSize.height/2);
}

public void actionPerformed(ActionEvent e)
{
  Object o;
  o=e.getSource();

  if(o==connect)
  {

    askForData();
    try
    {
      socket=new Socket(nameHost,port);
      toServer=new DataOutputStream(socket.getOutputStream());
      fromServer=new BufferedReader(new InputStreamReader(socket.getInputStream()));
      toServer.writeBytes("?NH?\n");
      askForObjName();
      rfs = new readFromServer();
      rfs.start();
    }
    catch(Exception e1){display.append(e1+"\n");}

  }
  if(o==disconnect)
  {
    showStatus.setEnabled(false);
    Switch.setEnabled(false);
    connection.setBackground(Color.RED);
    rfs.stop();
    try
    {
      toServer.close();
      fromServer.close();
      socket.close();
    }
    catch(Exception e2){display.append(e2+"\n");}
    display.append("Disconnected form server... \n");
    connection.setText("DISCONNECTED");
    connection.setBackground(Color.RED);
  }
  if(o==Switch||interruttore)
  {
  try
  {
    if(status)
    {
      write_datum((short)0x10);
      do_write();
      toServer.writeBytes("?0?\n");
      showStatus.setBackground(Color.RED);
      showStatus.setText("OFF");
```

```
        status=false;
      }
      else
      {
        write_datum((short)0x11);
        do_write();
        toServer.writeBytes("?1?\n");
        showStatus.setBackground(Color.GREEN);
        showStatus.setText("ON");
        status=true;
      }
      display.append("\nRequest sent to server\n");
    }
    catch(Exception e2){display.append(e2+"\n");}
          }
    if(o==exit)
    {
      System.exit(0);
    }
  }
}

public static void inizializeGraphics()
{
  frame=new myFrame("Client");
  c=frame.getContentPane();
  north=new JPanel();
  center=new JPanel();
  center_north=new JPanel();
  south=new JPanel();
  label=new JLabel("Client status: ");
  display=new JTextArea();
  connection=new JTextField("DISCONNECTED");
  connect=new JButton("connect");
  disconnect=new JButton("Disconnect");
  Switch=new JButton("Switch");
  showStatus=new JTextField("OFF");
  exit=new JButton("exit");
  connection.setBackground(Color.RED);
  display.setEditable(false);
  connection.setEditable(false);
  showStatus.setEnabled(false);
  Switch.setEnabled(false);
  c.setLayout(new BorderLayout());
  north.setLayout(new GridLayout(1,2));
  center.setLayout(new BorderLayout());
  center_north.setLayout(new GridLayout(1,3));
  connect.addActionListener(frame);
  disconnect.addActionListener(frame);
  Switch.addActionListener(frame);
  exit.addActionListener(frame);
  north.add(connect);
  north.add(disconnect);
  center_north.add(label);
  center_north.add(connection);
  center_north.add(Switch);
```

```
  center.add(center_north,BorderLayout.NORTH);
  center.add(new JScrollPane(display),BorderLayout.CENTER);
  south.add(showStatus);
  south.add(exit);
  c.add(north,BorderLayout.NORTH);
  c.add(center,BorderLayout.CENTER);
  c.add(south,BorderLayout.SOUTH);
  frame.setVisible(true);
}

public static void main(String args[]) throws Exception, InterruptedException
{
  status=false;
  lpt = new pPort();
  inizializeGraphics();
}

public static void askForData()
{
  String aux;
  aux=new JOptionPane().showInputDialog("Port number (press Enter for '2121'): ");
  if(!aux.equals(""))
  {
    try
    {
      port=Integer.parseInt(aux);
    }
    catch(Exception e){display.append("Wrong value, default value will be loaded(2121)..\n");}
  }
  aux=new JOptionPane().showInputDialog("Host name (press Enter for 'localhost'): ");
  if(!aux.equals(""))
  {
    try
    {
      nameHost=aux;
    }
    catch(Exception e){display.append("wrong value, dafult host name will be loaded (localhost)..\n");}
  }
}

public static void askForObjName()
{
  String aux;
  aux=new JOptionPane().showInputDialog("Please choose the object name : ");
  if(!aux.equals(""))
  {
    try
    {
      objName=aux;
      toServer.writeBytes(objName+"\n");
    }
    catch(Exception e56){display.append(e56+"\n");;}
  }
}

static class readFromPort extends Thread
```

```
{
                        public void run()
  {
    try
    {
      while(true)
      {
        do_read();
        if(Integer.toHexString(read_datum).equals("1"))  //turning OFF
        {
          write_datum((short)0x10);
          do_write();
          toServer.writeBytes("?0?\n");
          showStatus.setBackground(Color.RED);
          showStatus.setText("OFF");
          status=false;
          sleep(5000);
        }
        if(Integer.toHexString(read_datum).equals("0")) //turning ON
        {
          write_datum((short)0x11);
          do_write();
          toServer.writeBytes("?1?\n");
          showStatus.setBackground(Color.GREEN);
          showStatus.setText("ON");
          status=true;
          sleep(5000);
        }
      }
    }
    catch(Exception e){display.append(e+"\n");}
  }
}

static class readFromServer extends Thread
{
  String read;
  String rec_status;
  public void run()
  {
    while(true)
    {
      try
      {
        read = fromServer.readLine();
        if(read==null)
          break;
        else if(read.equals("??OK??"))
        {
          display.append("SL object connected!");
          Switch.setEnabled(true);
          showStatus.setEnabled(true);
          showStatus.setBackground(Color.RED);
          display.append("Connected to the server... \n");
          connection.setText("CONNECTED");
          connection.setBackground(Color.GREEN);
```

```
            write_datum((short)0x10);
            do_write();
            if(ck)
            {
              rfp = new readFromPort();
              rfp.start();
              ck=false;
            }
          }
          else if(read.equals("??EXISTING_NAME??"))
          {
            display.append("\nThe name of this object is already in the database, please select another one.\n");
            toServer.writeBytes("?NH?\n");
            askForObjName();
          }
          else if(read.equals("??CREATINGOBJ??"))
          {
            display.append("\nWaiting the connection of the SL object...");
            if(status)
              toServer.writeBytes("1\n");
            else
              toServer.writeBytes("0\n");
          }
          else if(read.equals("?PHP?"))
          {
            rec_status = fromServer.readLine();
            if(rec_status.equals("0"))
            {
              write_datum((short)0x10);
              do_write();
              showStatus.setBackground(Color.RED);
              showStatus.setText("OFF");
              status=false;
              display.append("Request received from the Server, now status is: "+status+"\n");
            }
            else if(rec_status.equals("1"))
            {
              write_datum((short)0x11);
              do_write();
              showStatus.setBackground(Color.GREEN);
              showStatus.setText("ON");
              status=true;
              display.append("Request received from the Server, now status is: "+status+"\n");
            }
          }
          else
            display.append("\nSomething went wrong, please try to connect later.\n");
        }
        catch(Exception e){display.append(e+"\n");}
      }
    }
  }
}
```

Listing 4.6: driver.java

4.10 Setting architecture

Next we need to understand that our architecture is a distributed system and show how Second Life and each client (“driver”) can run on different machines, and how they all communicate using the TCP/IP protocol. In addition we have to connect the php page with the middleware via sockets. For these reasons we need a fixed IP address for the machine where the middleware is running. Besides, if the client is not locally achievable it must have a fixed IP too. However, not all common users have a fixed IP address and therefore they could not use our architecture. For solving this problem we can use a service called Dynamic DNS. Dynamic DNS (DDNS) allows you to create a hostname that points to your dynamic IP address or URL.

Today, numerous providers, called Dynamic DNS service providers offer such technology and services on the Internet. They provide a software client program that automates this function. The client program is executed on a computer or device in the private network, it connects to the service providers sytems, and causes those system to link the discovered public IP address of the home network with a hostname in the domain name system. Depending on the provider, the hostname is registered within a domain owned by the provider or the customer's own domain name. These services can function by a number of mechanisms, often they use an HTTP service request since even restrictive environments usually allow HTTP service. This group of services is commonly also referred to by the term Dynaminc DNS, although it is not the standards-based DNS Update method. However, the latter might be involved in the providers systems.

In Microsoft Windows networks, Dynamic DNS is an integral part of Active Directory, because domain controllers register their network service types in DNS so that other computers in the Domain (or Forest) can access them.

For realizing our demo I have used the service offered by DynDNS. The image below shows the sowtware client program.

Figure 4.9: DynDNS sowtware client program

In this way both the clients and the middleware will be accessible from the “outside”. So we can modify the php page like this:

```
<?PHP

// the hostname where MySQL database is located
```

```
$ip = 'second-life.dnsalias.net';
$port = '2121';

?>
```

Listing 4.7: javaData.php

4.12 Further considerations and possible improvements

It would be possible to improve the readability of the code and optimize parts of our architecture. For example, one of the things we should improve are concerns related to safety and security. All the php pages should use POST method instead of GET. Obviously this means it is possible to modify the http requests made by the lsl script inside the SL object, and also the java lines that call these pages. We can assume that attacks from the virtual world against real life systems will become a realistic threat in the future. In addition, at the beginning, we have had another kind of difficulty. The real problem was delay in processing data going in and out of second life. In fact xml-rpc protocol was not well implemented in Second life, and it added a delay that I can estimate between 15 and 30 seconds. This was a real limitation because the connection between the two worlds has to happen in real time. In the last months, fortunately, The Second Life implementation of xml-rpc has significantly improved, and now the delay has reached the 2-3 seconds, so we can approximate this to a real time event flow.
Another possible improvement regards the solution of power electric problems. A good idea could be using relays. Here a possible electrical scheme:

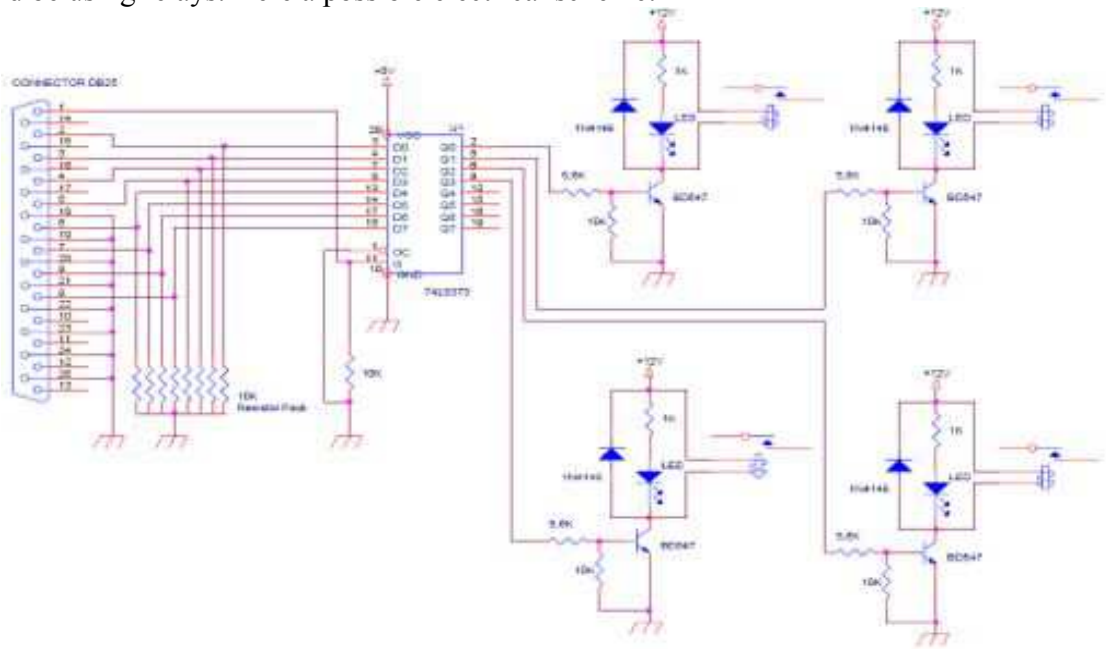


Figure 4.10: Possible improvements, a more complicated electrical scheme

The electronic circuit is very simple and it uses a single integrated circuit, 74L373. This is useful to buffer the four lines of the parallel port, to avoid overload. This integrated also presents another interesting feature: it can store, according to logical value applied to pin 11 (LE Latch Enabled), the levels of voltage applied to the inputs D0 ... D7. We can use this property to keep the relay excited according to the value that the parallel port provides, even when it is

disconnected. As you can see the pin 11 is connected to ground through a 10Kohm resistor. In this condition on outputs Q0 .... Q7 there will be the last read value on inputs D0 .... D7. To change the logic levels on the outputs Q0 .... Q7, we must provide to the pin 11 an high logical level(+5 V). In this way the parallel port's values will be transferred on the U1's outputs. Finally, the U1 will drive the four transistors that manage the relè.

My case study is very simple, and it wants only to describe a basic architecture for connecting Real and Virtual world; but these studies can be used as a starting point for important and more complicated projects.

Chapter 5
Conclusion and future developments

5.1 Conclusions

At the beginning of my training we did not know if a connection between real and virtual world was possible.We investigated the feasibility of connection and then we also evaluated the various options. Among these we have chosen the more flexible and open solution. We have built a generic architecture that permits us to manage new type of hardware by simply adding the "driver" without re-writing the whole software. Thanks to our generic architecture the connection between real and virtual world can be realized with any kind of hardware making it extremely flexible. Currently our middleware offers only the basic functions that are common to any hardware: add / delete, turn on / off, check. Nevertheless, to extend the capabilities offered by our middleware we will have just simply to add the desired functions. Anyway, anyone having even basic knowledge of informatics could easily understand why the whole problem can be reduced to an on/off event, between these two worlds. It does not matter what the application is that you want to realize, if I give you the functionality to allow you to transfer a string between Second Life and your own application, you can do whatever you want.

5.2 A more complicated demo

This is a lot of potential to create other more complex applications using our approach. To give you an idea of the great potential of our architecture I will discuss some work we started at the end of my placement period. We realized it only in part, but it is useful because it could be an important application, and it is also related to the second possible application, the virtual learning. Our intention is to keep the same architecture developed for the previous demo, but to use a more complicated hardware. The hardware we have chosen is usually used for test and measurement. We are going to look at connneting a waveform generator, a multimeter, and an oscilloscope using our system.

Figure 5.1: A waveform generator, a multimeter, and an oscilloscope

5.2.1 Background

We already had an existing software program made by a member of the University of Ulster to manage access and control the instrumention. The only problem is that this software is written in C#, instead of our middleware that is written in java.vAs you can imagine, there are many ways of connecting an oscilloscope to SL, and they often depend on what is your goal (and project background too). We simply made use of our software architecture and a tried and tested piece of software that we had. We have decided to address integration through Web Services to integrate applications on heterogeneous platforms developed. This way shows the remarkable flexibility of our architecture and the quite limitless development possibility.

Let us strat to explain what we have done.
We have read the managing software written in C#. So we also indentified the classes and their functions that control the hardware.
Here they are:

Oscilloscope

public class Agilent54622a

```
    /// Constructor. Connects to the instrumentation server and resets the instrument to its
        /// default state. Exceptions can be thrown here
        public Agilent54622a()

        /// Closes the session and disconnect from the instrument
        public void Close()

        /// Resets the instrument to its default state
        public CommandResult Reset()

        /// Autoscales the scope
        public CommandResult Autoscale()

        /// Reads the instrument
        public CommandResult ReadScope()

        /// Enables/disables channel 1
        public CommandResult SetChannel1(bool channelOn)

        /// Enables/disables channel 2
        public CommandResult SetChannel2(bool channelOn)

        /// Sets the volt base of the scope
        public CommandResult SetVoltBase1(string voltbase)

        /// Sets the volt base of the scope
        public CommandResult SetVoltBase2(string voltbase)

        /// Reads the volt base of the scope
        public string ReadVoltBase1()

        /// Reads the volt base of the scope
        public string ReadVoltBase2()

        /// Sets the time base of the scope
        public CommandResult SetTimeBase(string timebase)

        /// Reads the time base of the scope
        public string ReadTimeBase()

        /// Activates the measure channel frequency function on the scope
        public void Measure()
```

Listing 5.2.1: Agilent54622a.cs, class for managing the oscilloscope

Waveform generator

public class Hp33120a

```
/// Constructor. Connects to the instrumentation server and resets the instrument to its
    /// default state. Exceptions can be thrown here
    public Hp33120a()

    /// Closes the session and disconnect from the instrument
    public void Close()

    /// Resets the instrument to its default state
    public CommandResult Reset()

    /// Set the amplitude of the output signal
    /// <param name="volts">The amplitude in volts</param>
    public CommandResult SetAmplitude(double volts)

    /// Set the frequency of the output signal
    /// <param name="freq">The frequency in herz</param>
    public CommandResult SetFrequency(double freq)

    /// Set the waveform shape
    /// <param name="shape"></param>
    public CommandResult SetWaveFormShape(WaveFormShapes shape)

    /// Return the current configuration of the instrument
    public CommandResult GetAllSettings()

    /// Return the current amplitude in volts
    public double GetAmplitude()

    /// Return the current frequency in herz
    public double GetFrequency()

    /// Return the current waveform shape
    public WaveFormShapes GetWaveFormShape()
```

Listing 5.2.2: Hp33120a.cs, class for managing the Waveform generator

Multimeter

public class Hp33120a

```
/// Constructor. Connects to the instrumentation server and resets the instrument to its
        /// default state. Exceptions can be thrown here
        public Hp33120a()

        /// Closes the session and disconnect from the instrument
        public void Close()

        /// Resets the instrument to its default state
        public CommandResult Reset()

        /// Set the amplitude of the output signal
        /// <param name="volts">The amplitude in volts</param>
        public CommandResult SetAmplitude(double volts)

        /// Set the frequency of the output signal
        /// <param name="freq">The frequency in herz</param>
        public CommandResult SetFrequency(double freq)

        /// Set the waveform shape
        /// <param name="shape"></param>
        public CommandResult SetWaveFormShape(WaveFormShapes shape)

        /// Return the current configuration of the instrument
        public CommandResult GetAllSettings()

        /// Return the current amplitude in volts
        public double GetAmplitude()

        /// Return the current frequency in herz
        public double GetFrequency()

        /// Return the current waveform shape
        public WaveFormShapes GetWaveFormShape()
```

Listing 5.2.3: Hp33120a.cs, class for managing the Multimeter

5.2.2 Possible solution

Here is the software scheme modified for our study case:

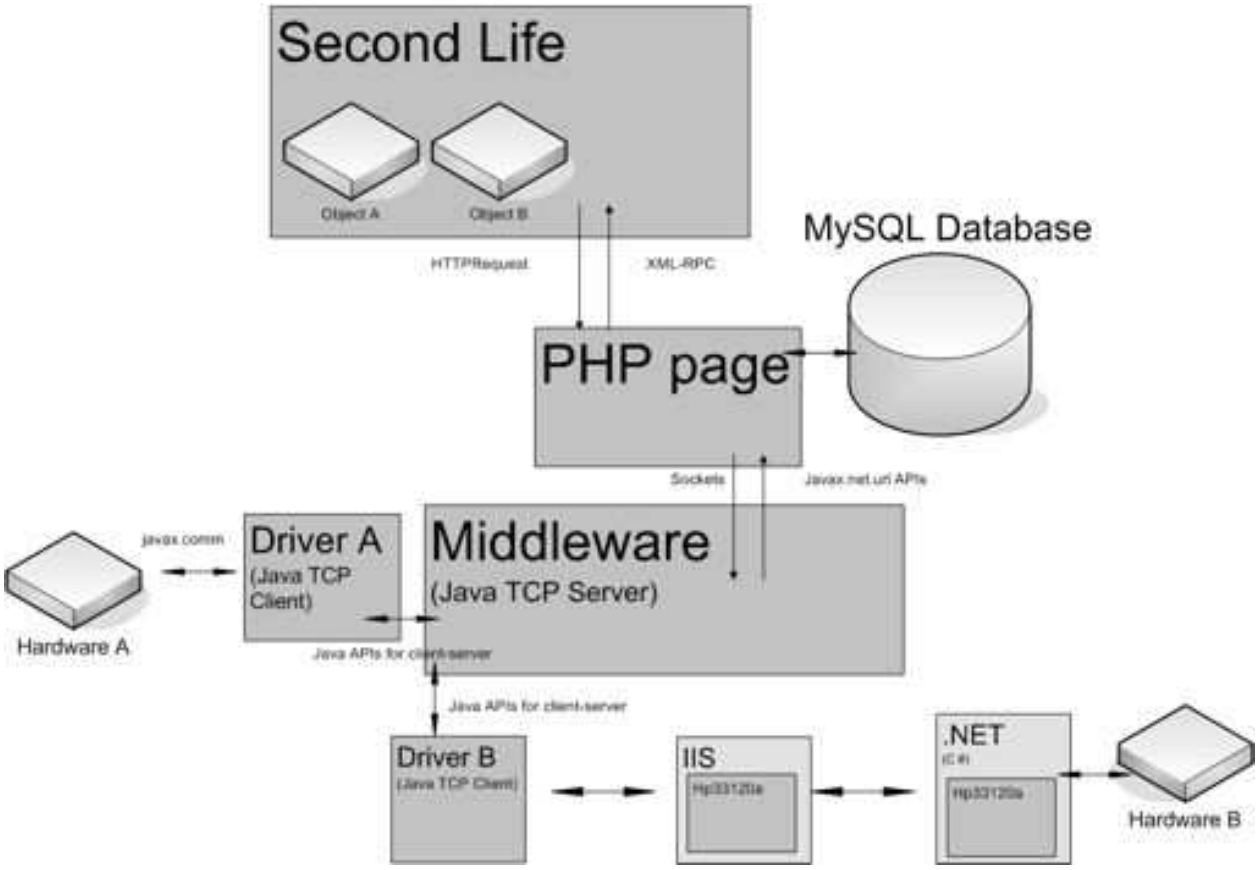


Figure 5.2: Modified software architecture

In short the service of the .NET application consists of the following .NET remote object (Hp33120aServant). It provides all the methods of Hp33120a.cs already developed.

```
using System;
public class Hp33120aServant : MarshalByRefObject
{
        public Hp33120aServant()
        {
                IRemoteHP33120a m_instrument = null;
                /// <summary>
                /// Constructor. Connects to the instrumentation server and resets the instrument
                to its default
                /// state. Exceptions can be thrown here
                /// </summary>
                public Hp33120a()
                {
                        if(ChannelServices.GetChannel("GTCPInstruments") == null)
                        {
                                IDictionary props = new Hashtable();
```

```
                props["Name"] = "GTCPInstruments";
                props["MaxTimeSpanToReconnect"] = "30000";
                props["PersistentConnectionSendPingAfterInactivity"] = "10000";
                props["ConnectTimeout"] = "30000";
                props["InvocationTimeout"] = "30000";
                GenuineTcpChannel channel = new GenuineTcpChannel(props,
                null, null);
                ChannelServices.RegisterChannel(channel);
        }
        m_instrument = (IRemoteHP33120a)
        Activator.GetObject(typeof(IRemoteHP33120a),
        "gtcp://diesel-iv:12345/Hp33120a.rem");
        m_instrument.StartSession();
}
/// <summary>
/// Closes the session and disconnect from the instrument
/// </summary>
public void Close()
{
        if (m_instrument != null)
        {
                m_instrument.EndSession();
        }
}
/// <summary>
/// Resets the instrument to its default state
/// </summary>
/// <returns></returns>
public CommandResult Reset()
{
        return m_instrument.Reset();
}
/// <summary>
/// Set the amplitude of the output signal
/// </summary>
/// <param name="volts">The amplitude in volts</param>
/// <returns></returns>
public CommandResult SetAmplitude(double volts)
{
        return m_instrument.SetAmplitude(volts);
}
/// <summary>
/// Set the frequency of the output signal
/// </summary>
/// <param name="freq">The frequency in herz</param>
/// <returns></returns>
public CommandResult SetFrequency(double freq)
{
        return m_instrument.SetFrequency(freq);
}
/// <summary>
/// Set the waveform shape
/// </summary>
/// <param name="shape"></param>
```

```
        /// <returns></returns>
        public CommandResult SetWaveFormShape(WaveFormShapes shape)
        {
            return m_instrument.SetWaveFormShape(shape);
        }
        /// <summary>
        /// Return the current configuration of the instrument
        /// </summary>
        /// <returns></returns>
        public CommandResult GetAllSettings()
        {
            return m_instrument.GetAllSettings();
        }
        /// <summary>
        /// Return the current amplitude in volts
        /// </summary>
        /// <returns></returns>
        public double GetAmplitude()
        {
            return m_instrument.GetAmplitude();
        }
        /// <summary>
        /// Return the current frequency in herz
        /// </summary>
        /// <returns></returns>
        public double GetFrequency()
        {
            return m_instrument.GetFrequency();
        }
        /// <summary>
        /// Return the current waveform shape
        /// </summary>
        /// <returns></returns>
        public WaveFormShapes GetWaveFormShape()
        {
            return m_instrument.GetWaveFormShape();
        }
    }
}
```

Listing 5.2.4: Hp33120aServant.cs

The remote object is made available by the following application server:

```
using System;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp;
public class Server
{
        public static int Main(string [] args)
        {
                System.Console.WriteLine("Starting ...");
                //It's listening on TCP port TCP 9999
                TcpChannel chan = new TcpChannel(9999);
                ChannelServices.RegisterChannel(chan, true);
                //And it shall record the object with a remote andpoint (logical name)
                //RemotingHp33120a in SigleCall mode (i twill create a new instance of the object
                for each request
                RemotingConfiguration.RegisterWellKnownServiceType(
                        Type.GetType("Hp33120aServant, object"),
                        "RemotingHp33120a",
                        WellKnownObjectMode.SingleCall);
                System.Console.WriteLine("Hit <enter> to exit...");
                System.Console.ReadLine();
                return 0;
        }
}
```

Listing 5.2.5: application server

The following commands are for the compilation of the .NET object:

```
csc        /out:..\bin\Hp33120aServant.dll        /debug+        /target:library
Hp33120aServant.cs
```

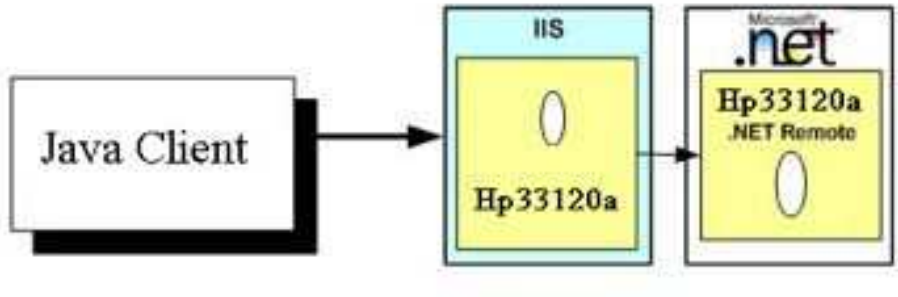
and for the compilation of the server application:

```
csc /debug+ /r:object.dll,System.Runtime.Remoting.dll server.cs
```

In this way we can obtaining the executable server.exe.

By the execution of server.exe all the . NET application's functions become remotely available.

Nevertheless, direct access to the .NET remote object it is not possible from other different platforms (eg J2EE, CORBA,…) because of the communication protocol.
How can allow direct access from a client B2B Java or Corba, since it is not possible between the J2EE application and. NET for the lack of interoperability of RMI / IIOP and Microsoft. NET communication protocol?
The answer is…. Web Services!



The Web Service is a C # class that uses decorator [WebMethod] to indicate ways to be exported as WebServices:

```
[WebService(Namespace="urn:Hp33120a.webservice.dotnet")]
public class Calculator:WebService
{
        [WebMethod]
        public Hp33120a()
        {
                try
                {
                        //When invoked, the WebService will retrieve the remote .NET object
                        Hp33120aServant obj = (Hp33120aServant)Activator.GetObject(
                                typeof(RemotingSamples.Hp33120aServant),"tcp://localhost:9999
                                /RemotingHp33120a");
                        if (obj != null)
                        {
                                //In this way we can invoke its methods
                                obj.Hp33120a();
                                ...

                        }
                ...
```

Practically, the service works like a "wrapper" of the remote. NET object.
Using the Axis WSDL2Java tool we obtained stub classes focusing on Web Service. NET endpoint:

```
java org.apache.axis.wsdl.WSDL2Java -o ..\src - p
```

This command allows to obtain the following java files: Hp33120a.java, Hp33120aLocator.java, Hp33120aSoap.java e Hp33120aSoapStub.java.

After that we have modified the java client in this way:

```
…
static URL endpoint
static Hp33120aSoap waveform_generator;
…
```

```
static class myFrame extends JFrame implements ActionListener
{
        public myFrame(String name)
        {
                super(name);
                setDefaultCloseOperation(EXIT_ON_CLOSE);
                Toolkit tk=Toolkit.getDefaultToolkit();
                Dimension screenSize=tk.getScreenSize();
                setBounds(screenSize.width/3,screenSize.height/3,screenSize.width/2,screenSiz
                e.height/2);
        }


        public void actionPerformed(ActionEvent e)
        {
                java.lang.Object o;
                o=e.getSource();
                if(o==connect)
                {
                        askForData();
                try
                {
                        socket=new Socket(nameHost,port);
                        toServer=new java.io.DataOutputStream(socket.getOutputStream());
                        fromServer=new                                  BufferedReader(new
                        InputStreamReader(socket.getInputStream()));
                        waveform_generator                          =                   new
                        Hp33120aLocator().getHp33120aSoap(endpoint);
...


public static void main(String args[]) throws Exception, InterruptedException
{
        status=false;
        try
        {
                URL endpoint = new java.net.URL(strEndpoint);
        }
        catch (Exception e) { e.printStackTrace();}
        inizializeGraphics();
}
...
```

5.2.3 Real time data visualization in Second Life

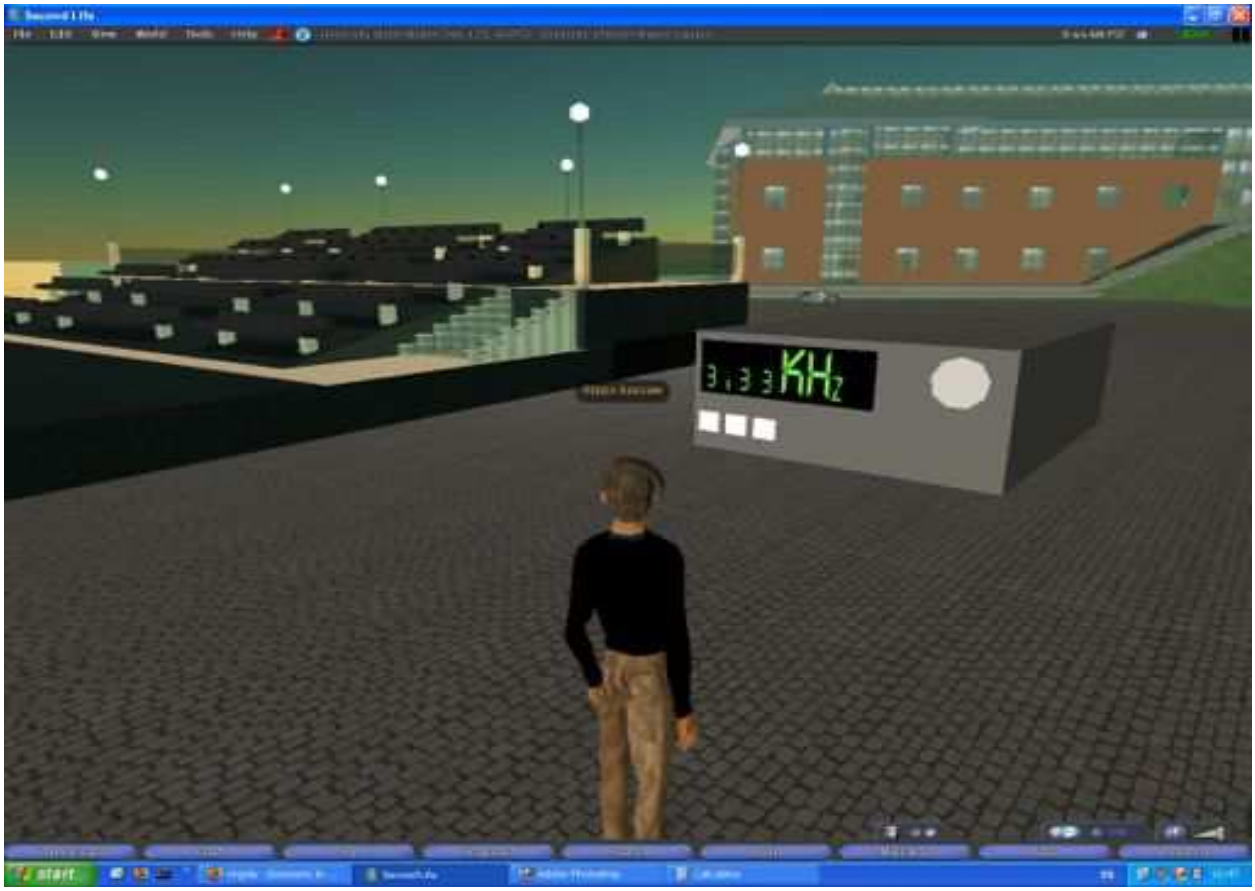


Figure 5.3: Real time data visualization in Second Life

We have created a new virtual object inside Second Life. Regarding the connection with the external world it's the same of our previous demo. Anyway we have modified the script for visualizing real time data. We have reproduced the instrument's LCD screen using an object for each ciphers. For every object that makes the screen it's possible to set the texture. For this reason we have uploaded 10 images (one for each decimal cipher).
Basically, when the virtual object recives data from the real world the "llSetTexture()" function is invoked on all the objects that make up the screen to show the numeric string received from the real world.

```
...
llSetTexture("1", ALL_SIDES);
...
```

5.2.4 Considerations

Regardless of how the solution is implemented, the very important thing to consider is the usefulness of this application. If you are able to visualize the oscilloscope output inside SL in almost real time, and you can manage the input from SL you can effectively control real world instrumentation remotely as if you were physically in front of it. If you can achieve this successfully then it is possible to use this in remote experimentation and virtual learning environments.

A key aspect of electronic engineering education is the application and evaluation of theory to real electronic circuits whereby students apply stimulus to circuits via instrumentation and monitoring the outputs. The use of real circuits and instrumentation enables the student to gain a better understanding of theoretical principles and the behavior of circuits under certain conditions, and also exposure to practical experience of instrumentation. Traditional remote lab environments provide two dimensional images or camera captures of such circuits and instrumentation however this does not provide the same student learning experience as the real lab environment where students can collaborate with other students and interact with circuits, and more importantly, are able to visualize phenomena such as current flow etc. Fundamentally, the environment provides a learning experience for students which more closely replicates real labs as it provides circuit animations complete with life-like function generator and oscilloscopes instruments, and demonstrates the opportunities afforded to students in supporting collaborative learning in Second Life. Future works aims to enable students to observe the dynamic flow of current in the circuit via user-defined frequencies on the output of the function generator. In addition more complex analog circuits consisting of transistors and op-amps will be explored.

5.3 Other possible applications: domotics

Using our architecture could be possible to realize applications for the home automation.
Given that the communication between the server and the clients is realized via sockets it is also possible to use the wireless technology. This is an important advantage because it is not necessary to cable all of a house which will save you a considerable amount of money. To do this we could realize a sort of box for each household appliance. In each box we could run Java SE for embedded systems so we can install a Java TCP Client. In addition it is possible to use the household appliances that you already have. It would be enough to add a different box (with the appropriate Java embedded TCP client) for each kind of appliance. It also could be possible to manage physical sensors (temperature, pressure, movement, fire, smoke, …) In this way, rooms can sense not only the presence of a person but also know who that person is and perhaps set appropriate lighting, temperature and music/TV taking into account day of week, time of day, and other factors. Other automated tasks may include setting the air conditioning to an energy saving setting when the house is unoccupied, and restoring the normal setting when an occupant is about to return.
More sophisticated systems can maintain an inventory of products, recording their usage through an RFID tag, and prepare a shopping list or even automatically order replacements (for example fridge).

Figure 5.4: Other possible applications: domotics

5.4 New technologies

The recent news about Second Life hacking give us a new idea for our project. In particular a very interesting rumour concerns a software tool called CopyBot that can clone avatars, objects and textures is stoking the ire of Second Life business owners who fear their creations could be illicitly copied and sold. CopyBot is just one application which can do the task of copying objects. It is a result of the people who are doing research on the libsecondlife project which aims at documenting the Second Life protocol. A protocol can be seen as a sort of language in which the client and server talk to each other. So in fact it is the deciphering of that data stream between them.
Additionally this project is an open source project which means that the code is freely available and can be modified and used by anybody as long as they do their work in terms of the license attached.

Here an example of CopyBot use:

Figure 5.5: An example of CopyBot use

So we have thought that another important possible way of exchanging data is developing a packet sniffing application.

Let us start to explain our new idea. As we said previously Second Life consists of clients and servers. On the server side there are first of all the servers which serve the sims (for each sim one server basically) and then there are some additional ones in the backend which handle e.g. login or provide the complete set of objects in existence (the asset server) which you can e.g. access via your inventory. Now in order to display objects and textures on your computer screen these need to be copied from the server to the client. In our understanding this will be a direct

connection between the sim server on which you are and your client while the sim server talks to the asset server if it needs additional objects or textures. The important point to note here is that all the objects (in the form of prim data) and textures (as bitmaps) will be transferred to your client which constructs the image from it by sending that data in a processed form to your graphics card (e.g. prim data will be converted into actual 3d objects constructed from points and vertices).
As this description implies it is always possible to take this data. And it happened already as with the GL intercept program or similar tools (only for textures though). In the same way we need to inject data into Second Life.

Here is the packet sniffing scheme:

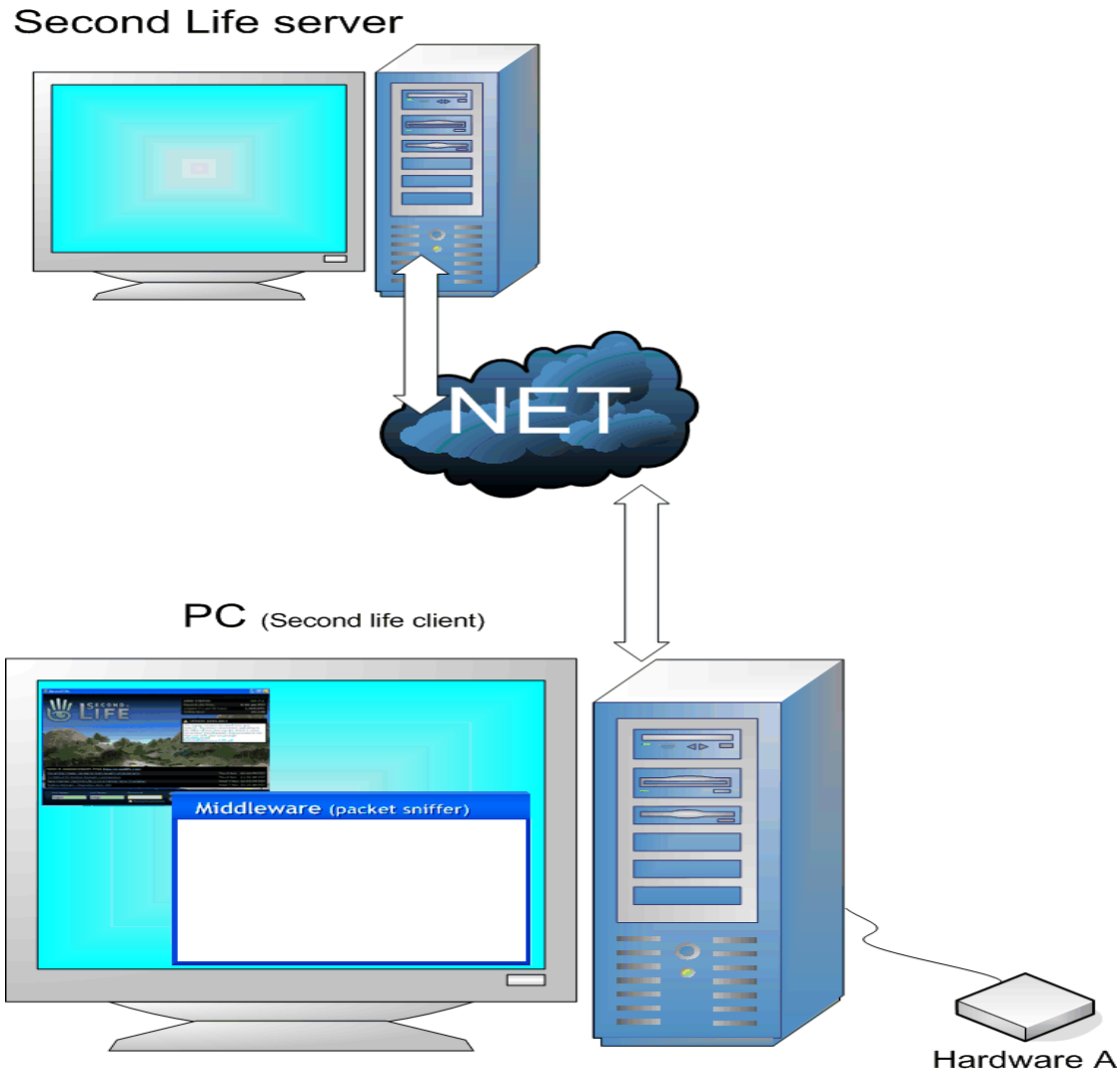
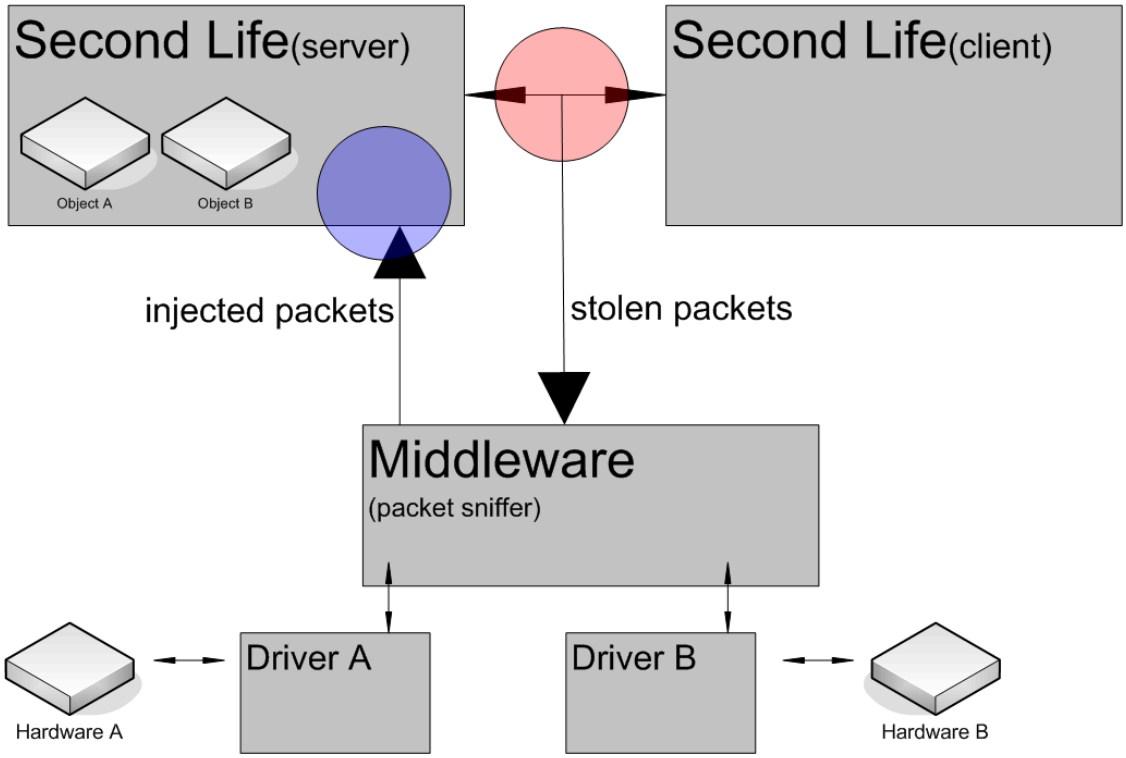


Figure 5.6: The packet sniffing scheme



To develop this new solution it is necessary to study the communication protocol between the Second Life's viewer and server. This is feasible because the client and the architecture are now open source.

Chapter 6
Acknowledgments

We want to say a big and real “THANK YOU!” to Michael Callaghan,who helped me during my project. He was really kind and helpful, and gave me everything I needed, overall he is a friend! I want to say thank you also to many other people of the “University of Ulster – Magee Campus” that helped me a lot, such as Jim Harkin, Shane Wilson, Malachy Mc Elholm, Neil Mc Donnell, and Edward McColgan. A last thank you goes obviously to the Leonardo program, that permitted me to have this experience, to the International relationship office of the University of Catania.

List of Figures

Listings

Bibliography

1) Book Betsy (2003), "What is a virtual world", *Virtual Worlds Review*, http://www.virtualworldsreview.com/info/whatis.shtml (retrieved on Dicember 2008)

2) Liedtke Volker (2004), "Second Life News", *second-life.com*, http://www.second-life.com/ (retrieved on Dicember 2008)

3) [Via Wikipedia] Mitch Wagner (March 5, 2007), "Inside Second Life's Data Centers", *InformationWeek*, http://en.wikipedia.org/wiki/Second_Life_Grid (retrieved on 2007-03-17)

4) [Via Secondlife.com] What is Second life, http://secondlife.com/whatis/ (retrieved on Dicember 2008)

5) MJ. Callaghan, J.Harkin, G.Scibilia, F.Sanfilippo, K. McCusker and S.Wilson (2008), "Experiential based learning in 3D Virtual Worlds", Intelligent System Research Center, University of Ulster, Northern Ireland

6) William Brogden (2007), "Web services and Second Life", *Search SOA*, http://searchsoa.techtarget.com/tip/0,289483,sid26_gci1255023,00.html (retrieved on Dicember 2008)

7) Seebach Peter (2007), "Hacking Second Life", Freelance writer, Plethora.net, http://www.ibm.com/developerworks/web/library/l-second-life-1.html (retrieved on Dicember 2008)

8) Douglas Beattie Jr. (2004), "Parallel port interfacing in Win32, using C/C++", Electronics R&D Engineering Consultant Ogden, Utah, U.S.A, http://www.hytherion.com/beattidp/comput/pport.htm (retrieved on Dicember 2008)

9) [Via Wikipedia] Sun Microsystems (1999), "Cross-platform", *Design Guidelines: Glossary*, http://en.wikipedia.org/wiki/Cross-platform (retrieved on Dicember 2008)

10) De Santis (2003), "Gestire quattro relè con la porta parallela del PC", desantix.it, http://www.desantix.it/index.php?page=show_articles&cmd=show_article&id=56&cookie=1 (retrieved on Dicember 2008)

11) Tao Takashi (2006), "The CopyBot controversy", taotakashi.wordpress.com, http://taotakashi.wordpress.com/2006/11/15/the-copybot-controversy/ (retrieved on Dicember 2008)