Scripted Avatars for the Virtual World of Second Life

Buchbauer Benjamin
b.buchbauer@student.tugraz.at

October 11, 2013

Contents

Abstract

Second Life is a 3D online virtual world where people from different physical locations can connect and meet other users in the same virtual location to interact with each other. The topic of this study is the link between the virtual world *Second Life* and the real world. We want to have a technical capability to quickly access the virtual world from everyone's computer without using a special 3D software. To do so we use a bot-based framework to connect to *Second Life* and create a web interface for the interaction with users. We implement a chat window where users can send and receive messages from the virtual world and access further data of users in the same location. In addition, the communication of all users will be stored in a database.

1 Introduction

A virtual world is a computer-based simulated 3D environment in which people from different physical locations can join and interact with each other any day. The state and the positions of every user and object in the specific region is saved, so everything remains the same if you join the virtual world again. Every user can move independently of one another, communicate via public and private chat, meet other users in arranged locations and use a multiplicity of objects with embedded artificial intelligence, like non-player characters, or with the simple purpose to embody your own virtual look. Nowadays virtual worlds play an important role to people's lives in addition to the real world. Because of that there is a lot of social media data and thus of interest to researchers in various areas, such as business, social science, communication, politics and education [7]. In this practical assignment we want to use a bot-based approach to connect the real world with the virtual world of *Second Life* due to the complicacy of the use of the *Second Life Viewer*, which is a provided 3D software to connect, explore and communicate in the virtual world. The 3D software has a high CPU usage and needs a very good graphic card to run smoothly. Hence we want to set up a web interface on which we can control the actions of our bot. We want to be able to visit different regions, get information about avatars and communicate via public and private chat without having to start up the *Second Life Viewer*.

The remainder of this report is organized as follows. Section 2 discusses the buildup of the virtual world of *Second Life*. Suitable ways to extract data from *Second Life* are presented in Section 3, which is followed in Section 4 by our approach to access the virtual world, extract data and display it on a website. Section 5 provides some final conclusions and directions for future work.

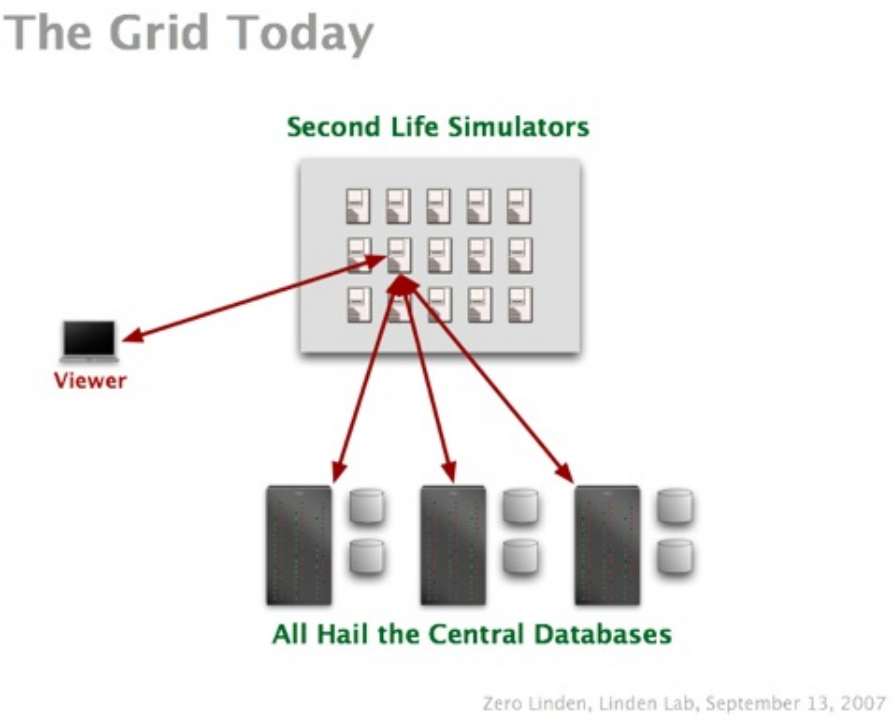
2 Second Life

Second Life is an online 3D virtual world owned by the company Linden Lab in which residents are called avatars. It was launched on June 23, 2003 and nowadays it is the leading platform for this kind of virtual worlds. In order to login, users first have to register with Linden Lab.

The *Second Life Viewer* is controlled by mouse and keyboard and avatars can interact with each other, play, explore the world, meet other avatars, participate in events, socialize, trade and create virtual property. To navigate through the virtual world there is an internal search engine to find different regions. Residents can teleport wherever they want to go and set landmarks to reach a specific region faster. In this way they can meet each other and communicate or trade objects, which are user built items, for example a wine glass as shown in the bottom left corner of Figure 2. They base upon a simple primitive shape and can either be placed somewhere on land or be attached to avatars. Communication can either be textual, which takes place in public or private chat, or vocally using *Second Talk*.

The world itself is simulated in the form of land surrounded by water. As shown in Figure 1, accessible regions are hosted by servers of Linden Lab, to which users are connected using the *Second Life Viewer*. This server farm is called the *Second Life Grid* and is a world which is divided into regions of 256x256 m areas of land. Each region has an unique name, a content flag (regions can be general, moderate or adult) and is simulated by a single named server instance. Regions without servers are represented as deep sea and cannot be flown over or entered. Residents can visit almost every region, unless the region has restricted access. In that case only authorized residents are able to enter that region. Avatars can also buy their own land and construct it however they wish. It must be purchased from a private seller, an auction or Linden Lab. In addition to membership fees (for example US$9.95/month premium membership and a granted 512 m^2 bonus lease) the cheapest monthly land use fee is US$5 for 1/128 region. To purchase an entire region you have to pay US$195 monthly. [3]

Figure 1: Second Life Grid (Linden Lab, 2007)



Furthermore every user has the ability to design the look of her avatar and live her desired life in the virtual world. This leads to the existence of various user groups that share different interests which is an important feature in *Second Life* to increase the social interaction among avatars. Moreover there are featured *Second Life* events and users can create community events. Such events feature a multitude of different activities, for example concerts, sports, education, different festivals or parties, art events, politics and so on. As shown in Figure 2

users have created a community event, in this case there is a party in a virtual jazz club, where people can dance and hang out. For *Second Life* residents this is the virtual counterpart to real life events.

Some universities use the virtual world of *Second Life* to conduct online lectures. Furthermore medical institutions are using *Second Life* simulations for training purposes where for example medical students are playing the role of doctors and are trying to solve clinical cases [5].

Figure 2: Party at Sweethearts Jazz Club

Second Life has an internal currency called Linden dollar (L$) that can be transferred into a real currency (US$) and vice versa. Therefore *Second Life* also has an internal economy. To get Linden dollar you have to use platforms as *LindeX* or *VirWox*, which are services that allow residents to trade L$ for US$. You can buy, sell, rent and trade with almost everything like land, vehicles, buildings, clothing, hair, skin, plants and so on. In addition you can engage different services like building, texturing, scripting, animating or let specific advertisement be displayed on another residents' land. On November 26, 2006 Anshe Chung has published a press release that she has become the first online personality to achieve a net worth exceeding one million US dollars from profits entirely earned inside a virtual world, *Second Life*. She began by selling and creating custom animations and later reinvesting the money to develop land which she has sold or rented to other users. [1]

Besides the above mentioned global information, there are some additional interesting details about the virtual world. *Second Life* recently celebrated its 10-Year Anniversary,

hence Linden Lab has published some facts about the current state of the virtual world.

- There are about 36 million residents registered, however round the clock there are usually 35000 - 60000 people online at once (many people have multiple accounts though)
- There have been US$3.2 billion total transactions among users for virtual goods within the *Second Life* economy
- More than 1 million users from around the world visit the *Second Life* world monthly
- There are about 1.2 million daily transactions for virtual goods
- The most purchased items are women's hairstyles
- There are 2.1 million user-created virtual goods for sale
- The *Second Life* landmass is nearly 700 square miles
- An average of about 400,000 new registrations are created monthly

All in all that are some astonishing facts about the current state of *Second Life*, especially the ongoing situation within the *Second Life* economy, where a big amount of transactions for virtual goods take place every day. [4]

Now we have heard about the fundamentals of the virtual world, on which our practical work is based on. In the following chapters we will go into more technical details.

3 How to Extract Data from Second Life

In the previous section we have given a quick overview about the virtual world of *Second Life* and in this section we will discuss the technical capabilities to extract data. There are two main methods, the *Linden Scripting Language* and the *Lib OpenMetaverse* testclient.

3.1 LSL - Linden Scripting Language

Residents of *Second Life* can embed scripts into objects to fulfill autonomous tasks. Scripts run inside objects and can be saved into the inventory of an avatar for backup purpose. It is also possible to put multiple scripts inside an object, they run synchronously.

LSL is a state-event driven scripting language which was created on the basis of a finite state automaton. It serves for the controlling and timing of objects, primitives and avatars in the virtual world. One can for example listen to chat channels, animate and control avatars, detect collision and get information about movement, primitives, regions and inventories.

A script starts once it is added to an object and turned on, they continue to run if a user is not logged in. Scripts are able to communicate with other objects and agents but it is

not possible that more than one event is detected concurrently thus there are no problems accessing global variables.

Unfortunately, there are several limitations in LSL as it contains over 300 library functions. Users can define additional functions and use native data structures like integer, float, strings, UUIDs, vectors and heterogeneous lists. However there are no arrays and there is no data storage. There is only maximum 16kB memory available to LSL scripts (mono scripts 64kB) but users can avoid the limitation by using multiple scripts (scripts are able to call other scripts). Moreover they are contributors to the server side lag, depending on their duration, the amount of work they are doing and if they are executing tasks continuously.

Data extraction using scripts is seriously affected by the aforesaid limitations of the *Linden Scripting Language*. To extract data script-enabled objects are used. They act as sensors in the *Second Life* world and can detect avatars and objects. Sensors log information such as position of the avatar/object and names of the animations that the avatars are currently playing. Unfortunately there are some problems associated with the use of sensors.

- Scripts can only be attached to objects. If a region does not allow the creation of scripted objects, then the scripted object should be worn by an avatar
- A sensor can detect only 16 avatars/objects
- The maximum sensor range is limited to 96 meters
- Use of more sensors leads to more server side lag
- Sensors can only record animation that are being played at the time of scanning (short animations may be missed)
- The recommended sensor repeating interval size is to the tune of minutes. This means a considerable amount of data is missed
- Because of the limitations in memory and LSL programming it is not possible to process complex data inside a script
- A HTTP response has to be started within the object so that it serves as a webserver. Then HTTP requests can be sent to the object to get information out of *Second Life* (maximum of 25 requests per 20 seconds)

[5, 2]

Below is an example of a simple *Hello World* script in LSL. When the script is activated, it starts in the default state *state_entry*. In this case *Hello World* is transmitted over channel 0, which is the public chat. Then there is another state *touch_start*, which is triggered every time a user touches the object, in which the script is embedded. In this particular example the script will output *Touched* in the public chat channel.

```
default
{
  state_entry()
  {
    llSay(0, "Hello World!");
  }

  touch_start(integer total_number)
  {
    llSay(0, "Touched.");
  }
}
```

In regard to our practical work there are some pros and cons for the use of the *Linden Scripting Language*:

Pros	Cons
Predefined function to listen to chat activity containing all necessary data. We don't need an extern server. We can put the script into an object in the virtual world which is running day in, day out.	Listening on channel 0 (public chat) can affect both script and sim performance. The *listen()* event will be triggered for every line of speech in regular chat. This should be avoided because it leads to significant server side lag. An object containing the script only senses chat activity in the surrounding area. It would be useless if most avatars reside on a completely different place. Data storage is very circumstantial in LSL, we first would have to transmit the data from the virtual world to the real world.

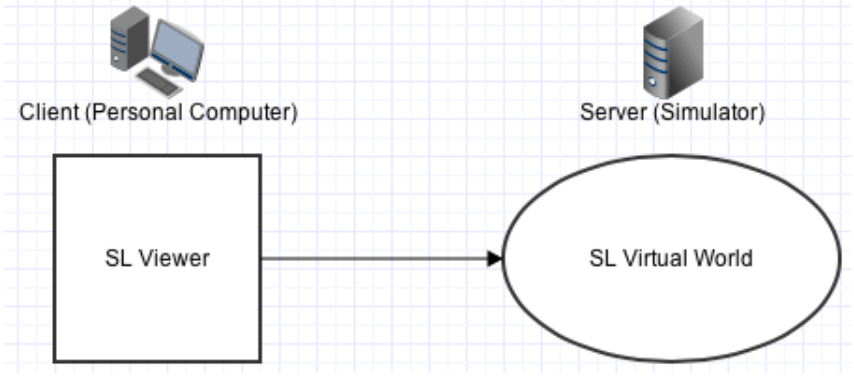
Table 1: Pros and Cons of LSL regarding to our Practical Work

3.2 Lib OpenMetaverse

Lib OpenMetaverse is a .NET based implementation of the communication protocol between client and server of 3D virtual worlds written in C#. The common usage, as shown in Figure 3, is to connect to the virtual world via *Second Life Viewer*. In contrast *Lib OpenMetaverse* is an open source project and contains a *Second Life* testclient, which is a command line client that uses the *Second Life* communication protocol.

Generally the same basic abilities as in the *Second Life Viewer* are available like running, flying, walking, chatting using a public chat channel or instant messaging and executing

Figure 3: Simple Overview of the Second Life Client-Server Architecture



animations such as clapping, laughing, dancing or crying. With the *Lib OpenMetaverse* testclient, in opposition to the *Second Life Viewer*, where the avatar is managed by mouse and keyboard, one has to control the avatar via instructions from the command line. Furthermore it is possible to sense and retrieve information about the surrounding environment. The *Lib OpenMetaverse* client receives updates whenever something changes in the environment such as messages in the public chat channel, instant messages, arrival of new avatars, avatar animations and velocity of an avatar/object. In summary the data received by the testclient can be compared to the events taking place in the visual field of an avatar controlled by a human.

Lib OpenMetaverse has increased flexibility compared to LSL as it can run as service without GUI, which means that the client runs automated and receives instructions. The user can expand the existing code with the needed functionality. Similar to the traditional client, one has to login with an existing *Second Life* account to connect to the virtual world. With the testclient one can connect multiple avatars at the same time to collect data faster and more efficiently. On the one hand we gather chat data and display it on the web browser and on the other hand we send chat data from the web browser to the virtual world. In contrast to LSL, where a script runs within an object, a users' avatar is her bot and she controls it with the command line on a user machine. There is no memory limit and no server side lag is introduced. Received data can be further processed and simply saved into a database. Despite the comfortable use of *Lib OpenMetaverse* there are some issues:

- It's not possible to extract accurate movement data.
- From the use of *Lib OpenMetaverse* to extract data from *Second Life* it follows that there is another avatar present in the environment which can be seen by other avatars and sensed by scripts. *Second Life* residents may find it annoying if they identify a bot which is acting as a sensor and just collecting data in a specific region. They may be distracted and confused and start complaining about it in the public chat. On the other

hand, the use of bots in *Second Life* is common and often used by land owners themselves so that there are more than zero avatars in the region and newly arrived avatars won't leave immediately. Approximately 5% of the entire *Second Life* population consists of bots. [6]

- There is no proper documentation of its design details and usability options available.

[5]

In regard to our practical work there are some pros and cons for the use of *Lib OpenMetaverse*:

Pros	Cons
No server side lag is introduced.	The testclient has to be running on an external server.
Fast and easy way to save data into output files and databases.	We have to login on every new access and execute the command to listen to chat activities.
More room for enhancements because of no memory limitation.	

Table 2: Pros and Cons of *Lib OpenMetaverse* regarding to our Practical Work

4 Accessing the Virtual World

In the previous section we have explained the possibilities of data extraction from *Second Life* and in this section we will discuss our approach to access the virtual world, how we extract data and how we display the gathered data. Due to the different natures of the two data extraction methods and their pros and contras, we have chosen to use *Lib OpenMetaverse* for our work because, instead of the high level complex information gained by LSL, we only need to access the easily recorded low level data of *Lib OpenMetaverse*.

Our goal is to create a webpage where users can login with their *Second Life* accounts and afterwards visit different regions. We want to have a small overview of the current location, have the public and private chat displayed and be able to look up profile data of nearby avatars.

4.1 Extracting Data from Second Life using Lib OpenMetaverse

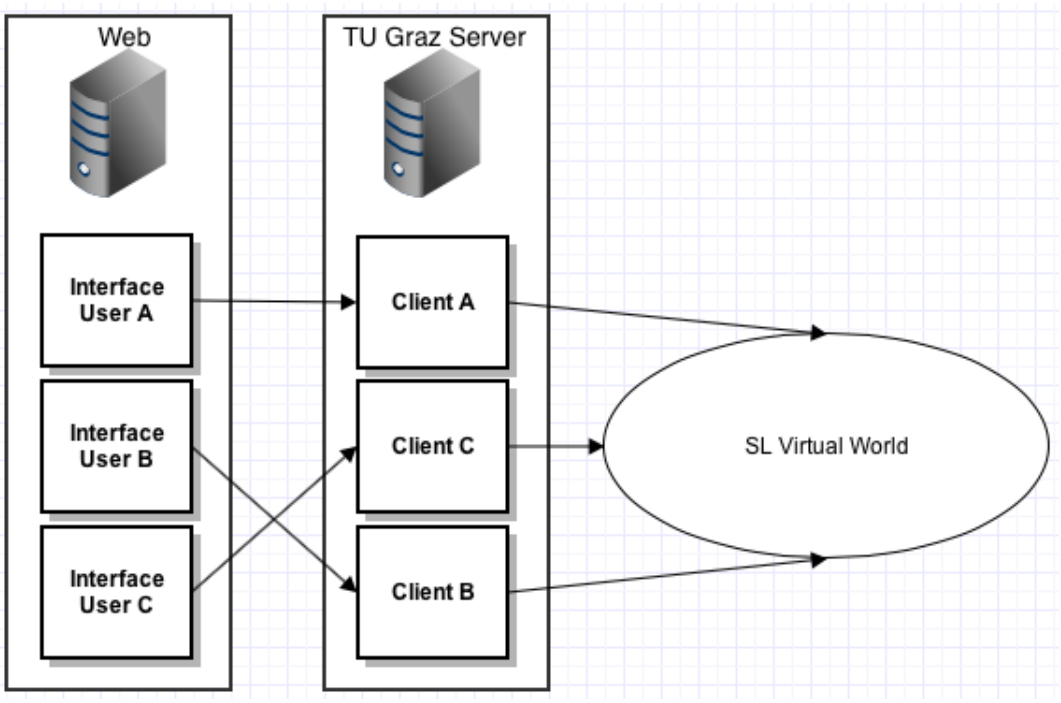
In this subsection we talk about the use of the *Lib OpenMetaverse* client from the start, which is initiated through the web interface, to the actual process of data extraction.

After entering username and password on the start page of our website and logging in, the compiled *Lib OpenMetaverse* testclient is started. It is located on a server of the Graz

University of Technology, as shown in Figure 4, with all the required files and directories. The login information is saved into a scriptfile which the client requires to connect to *Second Life*. It contains three commands, which the client will execute one after another during start-up. After that the client is ready to detect and execute further commands initiated by the web interface.

First we have the login command to authenticate with the *Second Life* server, which looks as follows *login firstname lastname password*. After that we need the command *waitforlogin* by what our client waits until all bots have succeeded or failed connecting to *Second Life*. Otherwise the client is not entirely ready and commands may be missed and not executed properly. Finally, we execute the command *filewatcher*, which is not part of the original framework but we have implemented by ourselves. An event handler is created that detects when an user input file has changed its content. Then the file data is read and the contained commands are executed. This is needed to handle different requests of the website which will be explained later.

Figure 4: Connection between the Web Interface, Lib OpenMetaverse Testclient and Second Life



If the login process has been successful, the next step for the users is to fill in the region they want to visit. As the input file gets modified, the event handler is called. In this case the input file contains the command *regionaccess region*, where region is the region name entered by the user on the webpage.

The client is built-on the *Command Pattern* and we simply added self-made commands

using the same course of action. As the name implies this command accesses the stated region.

First of all the existence of the region is verified. Meanwhile instant messages while being offline are retrieved and the teleportation to this area is prepared. To achieve the best possible future results, it is necessary to locate the coordinates of the place where at the moment most avatars remain. Then the bot is teleported to the stated region considering the located coordinates. However in *Second Life* there are certain teleport targets which means that the bot is not exactly landing on the desired coordinates. Because of that we try to reach this spot with a mix of manual walking and flying. As we can see in Figure 5, after the teleport sequence more commands are executed so that we can gather data and save it to the output file. Then the file will be read out and using *HTML5 Server Sent Events* the data will be sent to our web interface where the chat activity and the avatar profile data will be displayed.

We have implemented different commands like *regionaccess* and *filewatcher* but as example we want to discuss two of the most important functions of our work in the next part of the report.

4.1.1 Getting All Avatars in the Surrounding Area

By dint of the command *getavatars* data of every present avatar in the current region is fetched. It uses a timer so that the information is updated in a specific interval because avatars can enter or leave a region whenever they want to. The functionality is based on event delegation: the client sends several requests to the server and receives response packets. Then the related event is triggered and picked up packets are further processed. In this case the client is looking up every simulator we are currently connected to and moreover positions of all avatars in the effective simulator. Finally the client is trying to obtain every attendant avatar.

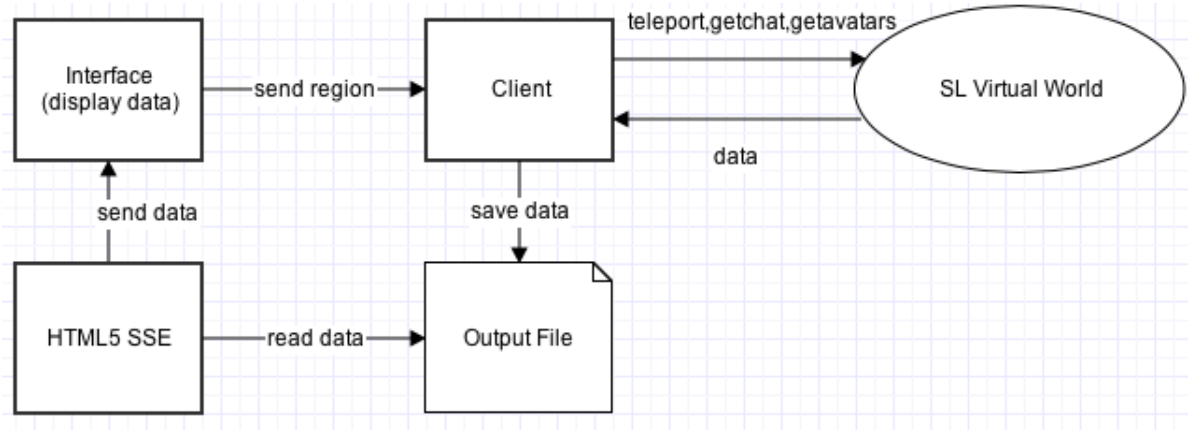
Primarily the names of the present avatars are needed to have a basic overview who resides in the current region. Furthermore his exact position in the region, his main group in the virtual world of *Second Life* and his UUID is looked up. The whole information is saved into an output file (see Figure 5) and refreshed in a specific time span stated in the timer so that the arrival or departure of avatars is noticed and immediately visible. If the user changes the visited region on the webpage, the bot teleports itself to the region and the simulator we are connected to also changes. That means that the client is directly fetching data from the new region and updating the ouput file accordingly.

Below is a code snippet of our command *getavatars*. We have a timer which executes the inner lines of code every twenty seconds so that our information is up-to-date. Afterwards we use delegates, which are like callback methods that are invoked when it is appropriate, to get the desired information step by step. We try to fetch every simulator we are connected to

and then get every avatar which is positioned in the current simulator. Afterwards we read out user data for the found avatars and save the required information to the output file.

```
System.Threading.Timer timer_ = new System.Threading.Timer(
  delegate(object sender)
  {
    Client.Network.Simulators.ForEach(delegate(Simulator simulator)
    {
      simulator.AvatarPositions.ForEach(delegate(KeyValuePair<UUID,
          Vector3> coarse)
      {
        Avatar foundAvatar = simulator.ObjectsAvatars.Find(delegate(Avatar
            avatar)
        { return (avatar.ID == coarse.Key); } );
        if (foundAvatar != null)
        {
          /* read out avatar information and save to output file  */
        }
      });
    });
  }, null, 0, 20 * 1000); //every 20 seconds
```

Figure 5: Structure with Commands



4.1.2 Listening to Chat Activity

The *getchat* command is working similarly to the above described *getavatars* command, except instead of fetching present avatars it is listening to communication between avatars. The data extraction is divided into the public chat channel and private instant messages but the basic concept is similar: the client requests the specific chat data and waits until the server responds. Depending on the type, the client either receives a *ChatEvent* packet or an

InstantMessage packet. From the *ChatEvent* packet we need to extract the simulator name, message, username, time, position of the avatar and chatflag. The chatflag distinguishes between shout, whisper, owner chat, region chat or normal say. Only the username and the chat message are written into the output file, as shown in Figure 5, however the whole information is inserted into a database where every type of communication is logged. The handling of the *InstantMessage* packet is exactly the same except instead of listening on public chat messages we listen on received instant messages.

The code below shows how we use the instant message delegate. When the method is invoked, we check if the type of the instant message packet is either *StartTyping* or *StopTyping*. In this case we discard the packet because it only notifies us that the user has started or stopped to type a message. We also check if the received packet is an instant message, which was sent by ourselves. If so we have to ignore it, otherwise we would handle the instant message twice, because of the original code of the framework for sending instant messages. If the packet fits our criteria, we save the instant message data to our output file and to our database.

```
1  Client.Self.IM += delegate(object sender, InstantMessageEventArgs e)
2  {
3    if (e.IM.FromAgentName != testClient.Self.Name && e.IM.Dialog !=
         InstantMessageDialog.StartTyping
4        && e.IM.Dialog != InstantMessageDialog.StopTyping)
5    {
6      try
7      {
8        string text = "(IM) from " + e.IM.FromAgentName + ": " +
             e.IM.Message;
9        string path = Environment.CurrentDirectory +
             "/client/bin/Userfiles/" + testClient.Self.FirstName + "/" +
             testClient.Self.FirstName + ".out";
10       using (System.IO.StreamWriter file = new
             System.IO.StreamWriter(path, true))
11       {
12         file.WriteLine(text);
13       }
14       string pos = Math.Round(e.IM.Position.X).ToString() + "/" +
             Math.Round(e.IM.Position.Y).ToString() + "/" +
             Math.Round(e.IM.Position.Z).ToString();
15       db.InsertMessage(text, e.IM.FromAgentName,
             DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss"), e.Simulator.Name,
             pos, "IM");
16     }
17     catch (Exception ex)
18     {
19       Console.WriteLine(ex.Message);
20     }
21   }
22 };
```

After we have fetched and saved all of the required information we have to send it to the web interface and represent it accordingly. The link between our client and our web interface will be discussed in the following subsection.

4.2 The Link between our Client and our Web Interface

The overall structure of our work is described in Figure 5. As we have mentioned above, the user starts by logging in on the website. If the user logs in for the first time with the web application, a new directory and additionally three important files are exclusively created for this user. This means that every user has his own login file, input file and output file. By using the login and input files, the client connects to the virtual world of *Second Life* and executes further commands, as soon as the user has entered the desired region which shall be looked up. As explained in the previous sections, we use the teleport function and our self implemented *getchat* and *getavatars* commands to fetch the needed data. Then the client saves every useful information into the correspondent output file, which stores all the data we want to be able to see on our website. How the data is displayed on our website will be explained in the next subsection.

4.3 Display of Data on a Website

As shown in Figure 6 our goal is to have seperate windows for the public chat and every single private chat with other avatars. The windows are organized by tabs and can be switched anytime. Furthermore we want to have a userlist, where users can start a private chat, show profile data and show the position of an avatar in the current region, which will be highlighted in a map.

The user starts by logging in with his *Second Life* account. If this account logs in for the first time, a new user directory and the required files are created. One of them is the login file, which contains the login data and commands that are immediately executed after the login process. After that the *Second Life* testclient is launched using the login input file. If the login is successful, the session is started. Otherwise the user is redirected to the start page. Next the user can enter the region she wants to visit. The region data is passed to the user-specific input file. The client notices that the file has changed, see Section 4.1, and executes the inserted command. In the meantime the website loads the corresponding map using the *Second Life Map API* so that the user gets an overview of the visited region. Generally speaking the website is divided into four parts, chat window, userlist, user profile data, map and can be found in Figure 6:

1 Chat window

Displays public and private chat and has a button to send a message in the active window with a dropdown menu to choose the chat type.

2 Userlist

List of present users in the current region.

3 User profile data

By clicking on a user of the userlist, the profile data is shown right next to the list.

4 Map

Gives an overview of the current region and highlights the position of an user.

If the user has entered a valid region, then a message stating the active user count of this region appears in the *Public Chat* tab. From that point on nearby chat and the userlist is shown in real-time using *Server Sent Events*. A php-file reads out every line of the user output file and sends the data to the website. Here an event handler for upcoming server-sent messages processes the information and adds the message to the chat window, appends an avatar to the userlist or refreshes the userlist.

The user is also able to communicate with other users in chat. First of all instant messages, which were sent while the user has been offline, are retrieved and displayed in the chat window. Then there is a dropdown-menu to choose the chat type from *Say*, *Shout* and *Whisper* for sending messages, with *Say* as standard chat flag. Right next to the chat window is the userlist of the active region. By double-clicking on an user or if an instant message is received, a new tab appears in the chat window. While such a tab is activated the user can send and receive private messages. Obviously it is possible to have multiple tabs opened at the same time. In this case, if a new message is received in a non-activated tab, the targeted tab is highlighted by changing the font color. So one can switch between tabs, add new tabs, close tabs and spot new messages immediately. To create the tab environment we used the *jQuery UI*. It is a curated set of user interface interactions, effects, widgets, and themes built on top of the *jQuery JavaScript Library*.[1] The following code snippet shows how we change back the tab font color to the standard value before it is activated.

```
1  $( "#tabs" ).tabs(
2  {
3          beforeActivate: function(event, ui)
4          {
5                  var active = ui.newTab.text().split(" ");
6                  active_tab = active[0] +" "+active[1];
7                  //change font color back to normal value
8                  changeColor(active_tab, "black");
9          }
10 });
11
12 function changeColor(tab_name, color)
13  {
```

[1] http://api.jqueryui.com

```
14          var i;
15          var tab_key = "";
16          for(i = 0; i < array_tabs.length; i++)
17          {
18                  if(array_tabs[i].name == tab_name)
19                  {
20                          tab_key = array_tabs[i].key;
21                          tabs.find( 'a[href="' +tab_key +'"]'
                                ).css('color', color);
22                          break;
23                  }
24          }
25  }
```

By single-clicking on an avatar in the userlist, a short table containing information about the user is shown right next to the list. Amongst others it shows the active group, the current position and the UUID of the avatar. In addition to that the subjacent map is updated and points out the position of the user via a small yellow dot. This is done by the means of *AJAX*, which allows web pages to be updated asynchronously, and the *Second Life Map API*. By clicking on a specific user, a *XMLHttpRequest* is created, which sends the data to a php script where further steps are initiated. The same thing happens with sent chat messages. In the case of the user profile data an *interpretRequest* event takes place because the php script sends back data containing information about the user profile. Thereby, the information is inserted into a table right next to the userlist. After that, the map is updated. This means that the iframe, which contains the map of the current region, is refreshed with the position of the selected user and is displayed with a so-called marker. Finally with a click on the *Logout* button the avatar logs off from the virtual world and the client is closed.

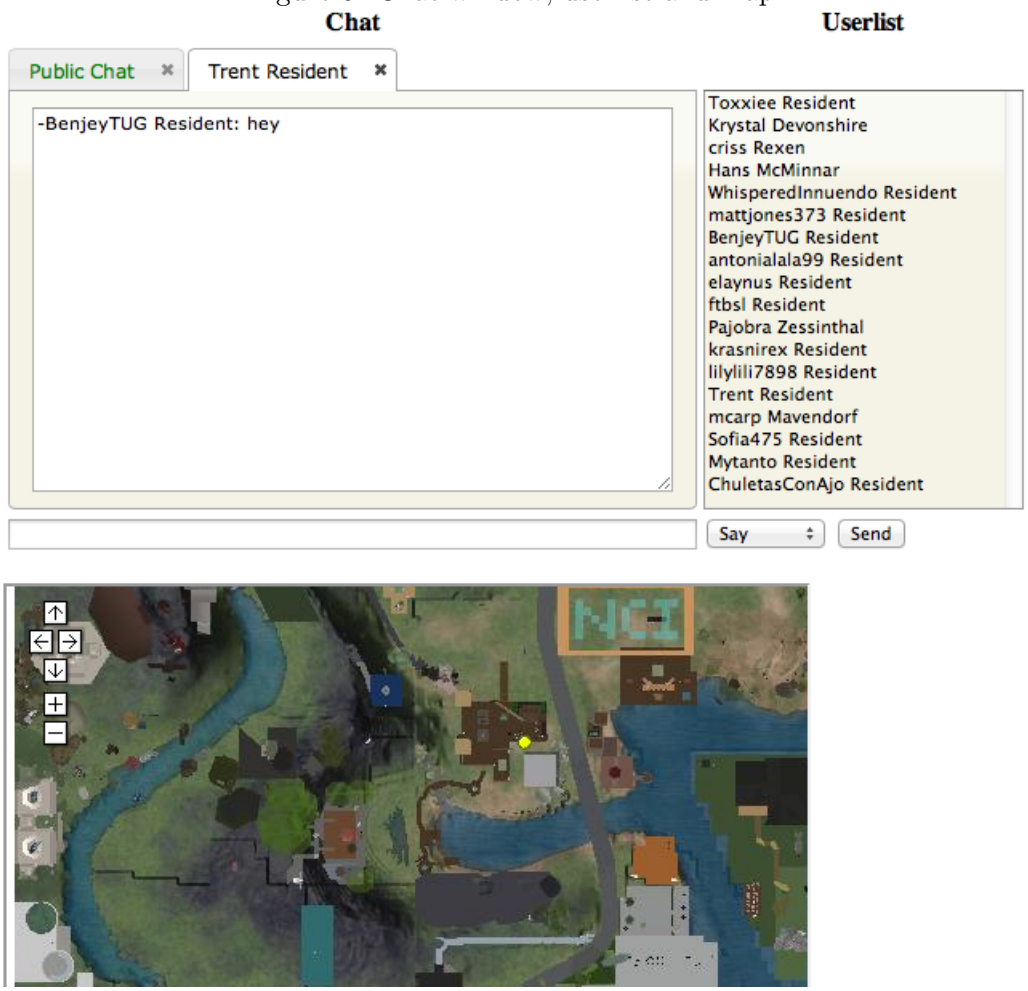
5 Conclusion and Future Work

In this work we tried to solve the problem of extracting data from *Second Life* using *Lib OpenMetaverse* to connect the virtual world and the real world. The project uses a bot based approach to collect avatar profile data and global communication data, which is stored in an output file. On the website the user is able to teleport to various regions, the chat data is displayed in real time and the profile of avatars can be looked up.

The success of the outcome is partly effected by the steadiness and speed of the internet connection. When having a very slow connection minor problems may occur, for example commands may be let out, like sent messages. Moreover it happened that the testclient sometimes needed varying time after the login sequence to be ready for further instructions. In this case a too early execution of the region lookup command results with an error when fetching the region data, although the entered region exists and is available.

All in all with our web interface we have created a serious alternative to the *Second Life*

Figure 6: Chat window, userlist and map



Viewer, with it's heavy capacity utilization of CPU and graphics card, primarily for users who want to have a quick look into the virtual world and see what is going on. It is very useful because we have implemented a fast way to access different regions, have a conversation in public chat with multiple avatars and start a private chat with whoever one wants. Because of that it is perfect for people who don't have a sufficient computer at one's disposal or just don't have much time, to use our web interface and have a look into the virtual world of *Second Life*.

In future work the service will be open to the public and we will have a look at the communication data which is stored in our database and analyze it thoroughly regarding frequency of usage and avatar behavior.

References

[1] Anshe Chung. Anshe chung becomes first virtual world millionaire. http://www.anshechung.com/include/press/press_release251106.html, November 2006. 5

[2] Robert J Cox and Patricia S Crowther. A review of linden scripting language and its role in second life. In *Computer-Mediated Social Networking*, pages 35–47. Springer, 2009. 7

[3] LindenLab. Mainland pricing and fees. http://secondlife.com/land/pricing.php, 2013. 4

[4] LindenLab. Second life celebrates 10-year anniversary. http://lindenlab.com/releases/second-life-celebrates-10-year-anniversary, June 2013. 6

[5] Surangika Ranathunga, Stephen Cranefield, and Martin Purvis. Extracting data from second life. 2011. 5, 7, 10

[6] Matteo Varvello, Stefano Ferrari, Ernst Biersack, and Christophe Diot. Exploring second life. *IEEE/ACM Transactions on Networking (TON)*, 19(1):80–91, 2011. 10

[7] Yulei Zhang, Ximing Yu, Yan Dang, and Hsinchun Chen. An integrated framework for avatar data collection from the virtual world: A case study in second life. 2010. 3

List of Figures

List of Tables