Web services for the management of persistent online game factions

Directeur de mémoire:
Pr. Esteban Zimányi

Co-promoteur de mémoire:
Ir. Jehan Snyers d’Attenhoven

Mémoire de fin d’études présenté par
François Deliège

en vue de l’obtention du grade
d’Ingénieur Civil Informaticien
Web services for the management of persistent online game factions

Final paper for the degree of
Civil Engineer specialized in Computer Science

Author: François Deliège
Advisor: Pr. Esteban Zimányi
Co-advisor: Ir. Jehan Snyers d’Attenhoven
Abstract

Star Wars Combine is a game involving thousands of players in a virtual world. Each player impersonates a character that continues evolving even when the player is not connected. Players have formed groups, called factions, that are self-organized. The more members a faction has, the more complicated to manage it becomes.

The goal of this work was to create an infrastructure to allow faction management tools to automatically update their data with the information maintained on the game server. The web services technology was chosen for the various advantages it offers. The new component has been implemented successfully and has been appreciated by many faction tool developers.

While this technology is already widely accepted in the business world, its use in a game context is totally new. It seems that web services have gained their place in the game world and will continue to be developed. Hopefully, this experiment will convince other game designers to adapt their platforms in a similar manner.
Acknowledgements

I'd like to thank...

My advisor, Professor Esteban Zimányi, and my co-advisor, Jehan Snyers d’Attenhoven, without whom this passionating work wouldn’t have been possible;

Professor Alf Weaver who asked me to lead a project for developing and integrating secure web services into the HIPAA distributed system;

Puyan Dadvar who helped me understanding the web services security stack and setting up the public key infrastructure required;

All the developers of the Star Wars Combine for being so helpful and the awesome achievements they have realized so far with the Combine;

My parents and my far away little sister;

Marie-Ange, Celine, Marie, Alana, Laure, Aurelie, Kyle, Inès, Sylvie, Emeline, Nike, Audrey, Jenny, Sarah, Jente, Florence, Gersende, Geraldine, Fulvia, Anna-Maria, Balais, Caroline, Charlotte, Elena, Julie, Maude, Natalie, Severine, Sophie, Van, Leona, Jean, Mike, Nicolas, Julien, Sam, Denis, Jacopo, Guyguy, Fred, Yves, Steph, Xa, Raf, Guillaume, Ced, Paul le pécheur, Matthias, Quentin, Ben le paki, Arnaud, Remi, Christophe, Michou, Pef and all the ones I have forgotten to mention.

Cheers,

–Francois
Contents

Abstract i

Acknowledgements ii

Contents iii

Figures vi

1 Introduction 1
   1.1 Context .................................................. 1
   1.2 Objectives ............................................... 2
   1.3 The approach ............................................ 3

2 Massively multi-player on-line role-playing games 5
   2.1 What is a MMORPG? ........................................ 5
   2.2 Browser-based MMORPGs .................................. 7
   2.3 Star Wars Combine overview ............................... 10
   2.4 Evolution of the SWC client-server architecture ....... 12
      2.4.1 Today : direct user interaction ....................... 12
      2.4.2 Tomorrow: direct trust relationship ................. 13
      2.4.3 The day after tomorrow: indirect trust relationship . . 14

3 Web services 17
   3.1 What are web services? .................................. 17
   3.2 Why to use web services? ................................. 18
   3.3 Web services in the business world ..................... 19
   3.4 Web services in games .................................. 21
## CONTENTS

### 4 Technological overview

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 The web services layers and interoperability stack</td>
<td>24</td>
</tr>
<tr>
<td>4.1.1 Layers</td>
<td>24</td>
</tr>
<tr>
<td>4.1.2 The transport layer</td>
<td>25</td>
</tr>
<tr>
<td>4.1.3 The messaging layer</td>
<td>26</td>
</tr>
<tr>
<td>4.1.4 Accessing web services with SOAP</td>
<td>27</td>
</tr>
<tr>
<td>4.1.5 Describing web services with WSDL</td>
<td>30</td>
</tr>
<tr>
<td>4.1.6 Finding web services with UDDI</td>
<td>31</td>
</tr>
<tr>
<td>4.1.7 Orchestrating web services with WSFL</td>
<td>33</td>
</tr>
<tr>
<td>4.2 Securing web services</td>
<td>34</td>
</tr>
<tr>
<td>4.2.1 Security requirements</td>
<td>34</td>
</tr>
<tr>
<td>4.2.2 SSL</td>
<td>35</td>
</tr>
<tr>
<td>4.2.3 XML signature</td>
<td>35</td>
</tr>
<tr>
<td>4.2.4 XML encryption</td>
<td>36</td>
</tr>
<tr>
<td>4.2.5 SAML</td>
<td>37</td>
</tr>
<tr>
<td>4.2.6 XACML</td>
<td>39</td>
</tr>
<tr>
<td>4.2.7 WS-Security</td>
<td>39</td>
</tr>
<tr>
<td>4.2.8 Other web services security specifications</td>
<td>40</td>
</tr>
</tbody>
</table>

### 5 Solution design

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1 Specifications</td>
<td>42</td>
</tr>
<tr>
<td>5.2 Web services motivation</td>
<td>43</td>
</tr>
<tr>
<td>5.3 Technical constraints</td>
<td>44</td>
</tr>
<tr>
<td>5.3.1 Server environment</td>
<td>44</td>
</tr>
<tr>
<td>5.3.2 Clients</td>
<td>45</td>
</tr>
<tr>
<td>5.4 Libraries</td>
<td>45</td>
</tr>
<tr>
<td>5.4.1 Choosing the library</td>
<td>45</td>
</tr>
<tr>
<td>5.4.2 Toolkit</td>
<td>47</td>
</tr>
<tr>
<td>5.5 SWC design evolution</td>
<td>49</td>
</tr>
<tr>
<td>5.5.1 SWC current design</td>
<td>49</td>
</tr>
<tr>
<td>5.5.2 SWC web services design</td>
<td>50</td>
</tr>
<tr>
<td>5.6 Security design overview</td>
<td>52</td>
</tr>
<tr>
<td>5.7 Authentication</td>
<td>54</td>
</tr>
<tr>
<td>5.7.1 Connecting to a public web service</td>
<td>55</td>
</tr>
<tr>
<td>5.7.2 Authentication with web services</td>
<td>56</td>
</tr>
</tbody>
</table>
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Context overview</td>
<td>2</td>
</tr>
<tr>
<td>2.1</td>
<td>MMORPG examples</td>
<td>6</td>
</tr>
<tr>
<td>(a)</td>
<td>EverQuest</td>
<td>6</td>
</tr>
<tr>
<td>(b)</td>
<td>World of Warcraft</td>
<td>6</td>
</tr>
<tr>
<td>(c)</td>
<td>Dark Age Of Camelot</td>
<td>6</td>
</tr>
<tr>
<td>2.2</td>
<td>Daimonin, a free MMORPG</td>
<td>7</td>
</tr>
<tr>
<td>2.3</td>
<td>Kings of chaos race selection page</td>
<td>8</td>
</tr>
<tr>
<td>2.4</td>
<td>Legend of the Green Dragon</td>
<td>8</td>
</tr>
<tr>
<td>2.5</td>
<td>KOL logo</td>
<td>9</td>
</tr>
<tr>
<td>2.6</td>
<td>Runescape</td>
<td>9</td>
</tr>
<tr>
<td>2.7</td>
<td>A SWC character</td>
<td>11</td>
</tr>
<tr>
<td>2.8</td>
<td>A SWC faction</td>
<td>11</td>
</tr>
<tr>
<td>2.9</td>
<td>Client-server model</td>
<td>12</td>
</tr>
<tr>
<td>2.10</td>
<td>Direct trust model</td>
<td>13</td>
</tr>
<tr>
<td>2.11</td>
<td>Indirect trust between portals</td>
<td>15</td>
</tr>
<tr>
<td>2.12</td>
<td>Indirect trust between clients</td>
<td>15</td>
</tr>
<tr>
<td>3.1</td>
<td>Web services in the industry</td>
<td>20</td>
</tr>
<tr>
<td>(a)</td>
<td>Google web services API</td>
<td>20</td>
</tr>
<tr>
<td>(b)</td>
<td>Amazon web services API</td>
<td>20</td>
</tr>
<tr>
<td>3.2</td>
<td>Tic Tac Toe game using web services</td>
<td>22</td>
</tr>
<tr>
<td>3.3</td>
<td>Chrysler game used for filling CRM databases</td>
<td>22</td>
</tr>
<tr>
<td>4.1</td>
<td>Web services layers</td>
<td>24</td>
</tr>
<tr>
<td>4.2</td>
<td>Web services scenario</td>
<td>25</td>
</tr>
<tr>
<td>4.3</td>
<td>Structure of a SOAP message</td>
<td>28</td>
</tr>
<tr>
<td>4.4</td>
<td>The Model View Controller paradigm</td>
<td>30</td>
</tr>
</tbody>
</table>
**LIST OF FIGURES**

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.5</td>
<td>WSDL structure</td>
<td>30</td>
</tr>
<tr>
<td>4.6</td>
<td>Dynamic communication with WSDL</td>
<td>31</td>
</tr>
<tr>
<td>4.7</td>
<td>UDDI architecture</td>
<td>32</td>
</tr>
<tr>
<td>4.8</td>
<td>UDDI registry conceptual view</td>
<td>32</td>
</tr>
<tr>
<td>4.9</td>
<td>UDDI role</td>
<td>33</td>
</tr>
<tr>
<td>4.10</td>
<td>Recursive composition of web services</td>
<td>34</td>
</tr>
<tr>
<td>4.11</td>
<td>Protection at every point</td>
<td>35</td>
</tr>
<tr>
<td>4.12</td>
<td>XML digital signature</td>
<td>36</td>
</tr>
<tr>
<td>4.13</td>
<td>XML encryption</td>
<td>37</td>
</tr>
<tr>
<td>4.14</td>
<td>SAML producer-consumer model</td>
<td>38</td>
</tr>
<tr>
<td>4.15</td>
<td>Single Sign On concept</td>
<td>38</td>
</tr>
<tr>
<td>4.16</td>
<td>XACML model</td>
<td>39</td>
</tr>
<tr>
<td>4.17</td>
<td>Roadmap of web services security specifications</td>
<td>40</td>
</tr>
<tr>
<td>5.1</td>
<td>NuSOAP Graphical User Interface</td>
<td>46</td>
</tr>
<tr>
<td>5.2</td>
<td>NuSOAP WSDL auto generated document</td>
<td>47</td>
</tr>
<tr>
<td>5.3</td>
<td>Client mechanism</td>
<td>48</td>
</tr>
<tr>
<td>5.4</td>
<td>Server mechanism</td>
<td>49</td>
</tr>
<tr>
<td>5.5</td>
<td>Darkness design overview</td>
<td>50</td>
</tr>
<tr>
<td>5.6</td>
<td>Web services server components</td>
<td>51</td>
</tr>
<tr>
<td>5.7</td>
<td>Layers with web services</td>
<td>52</td>
</tr>
<tr>
<td>5.8</td>
<td>Design Overview</td>
<td>53</td>
</tr>
<tr>
<td>5.9</td>
<td>Security mechanism overview</td>
<td>53</td>
</tr>
<tr>
<td>5.10</td>
<td>Connection to a public web service</td>
<td>55</td>
</tr>
<tr>
<td>5.11</td>
<td>Login procedure</td>
<td>57</td>
</tr>
<tr>
<td>5.12</td>
<td>Reusing a security token</td>
<td>58</td>
</tr>
<tr>
<td>5.13</td>
<td>Logout procedure</td>
<td>59</td>
</tr>
<tr>
<td>5.14</td>
<td>Rules management tool</td>
<td>61</td>
</tr>
<tr>
<td>5.15</td>
<td>Authorization rules for WS requiring client authentication</td>
<td>63</td>
</tr>
<tr>
<td>5.16</td>
<td>Authorization rules for WS requiring user authentication</td>
<td>63</td>
</tr>
<tr>
<td>5.17</td>
<td>Reading the News with the PHP client</td>
<td>66</td>
</tr>
<tr>
<td>5.18</td>
<td>Integration in Eclipse 3.1</td>
<td>66</td>
</tr>
<tr>
<td>5.19</td>
<td>C# client</td>
<td>67</td>
</tr>
<tr>
<td></td>
<td>(a) Login page</td>
<td>67</td>
</tr>
<tr>
<td></td>
<td>(b) Inventory Listing</td>
<td>67</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>--------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>5.20</td>
<td>Integration in Visual Studio 2003</td>
<td>67</td>
</tr>
<tr>
<td>6.1</td>
<td>Galactic Empire news system</td>
<td>69</td>
</tr>
<tr>
<td>6.2</td>
<td>CenterPoint Space Station inventory</td>
<td>70</td>
</tr>
<tr>
<td>6.3</td>
<td>Center Point dynamic galactic map</td>
<td>70</td>
</tr>
<tr>
<td>6.4</td>
<td>Web services usage</td>
<td>71</td>
</tr>
<tr>
<td>A.1</td>
<td>Setting user permissions</td>
<td>79</td>
</tr>
<tr>
<td>A.2</td>
<td>Removing an auto-grant</td>
<td>80</td>
</tr>
<tr>
<td>B.1</td>
<td>CSS example</td>
<td>86</td>
</tr>
<tr>
<td>C.1</td>
<td>Creating a rule</td>
<td>91</td>
</tr>
<tr>
<td>C.2</td>
<td>Deleting a permission</td>
<td>91</td>
</tr>
<tr>
<td>C.3</td>
<td>Packages management</td>
<td>91</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Context

The Star Wars Combine, briefly SWC, is defined by its main developers as follows:

“The Star Wars Combine is a free massively multi-player on-line role-playing simulation game, based on the Star Wars universe, developed by amateurs during their spare time.”

Through its simulation aspects, SWC introduces and develops the faction concept. Factions are groups of players who are given in-game tools to organize and manage themselves. The SWC faction concept is different from the guild concept usually used in game design references. The possibilities offered to the SWC factions are much larger.

Through the game interface, the faction leaders have access to various data concerning their factions: membership statistics, events and transactions reports, inventory status, maps, etc. With time, a growing need emerged for the factions to be able to retrieve information in a usable format and to import it in their own application.

The idea is to develop a client-server architecture using web services technology. By invoking remote methods, factions will be able to perform various actions without using the game interface anymore, but by using their own application instead. However, no assumption can be made about the client programming language, or its platform or network environment.
Among the system specifications, the communication system to be created will have to be easy for both server and client programmers to implement, while offering security features to allow users to authenticate themselves and retrieve the data in relation to their characters’ access levels.

1.2 Objectives

The four objectives of this work are:

1. **To imagine a web services architecture to fit the application needs:**
   Once the game mechanisms are understood, it is necessary to define specific application needs and constraints. In order to imagine a new architecture, drafts need to be prepared and circulated for their review amongst the game community experts. It is obvious that the architecture will be inspired by other technological standards and will need to remain open to future developments.

2. **To integrate the architecture into the already deployed production server:**
   Integration requires a full adaptation with the production server by respecting its design and architecture. During deployment, special attention needs to be paid to avoiding both server crash or security holes creation. Security issues imply the respect of both server security and game rules.
3. **To demonstrate the technology usability and added value:**

   This very important step, which requires more social engineering than proper computer skills, also implies “evangelist” aptitudes. Prototypes to demonstrate both the feasibility and the usefulness of the new component need to be created.

4. **To measure the technology acceptance among the whole community:**

   Various elements will need to be taken into consideration: number of uses, questions asked, requests for new features, etc. The objective is to provide the game administrators with enough data to help them making the decision whether or not to continue to develop web services technology within the game.

Along with these objectives come documentation and community involvement obligations to insure long-term continuity to the project.

### 1.3 The approach

This thesis intends to explain how a web services architecture could be added to a game server already in place. It is structured as follows.

Chapter 2 introduces massively multi-player role-playing games and persistent virtual worlds along with some well-known examples. It describes the basic concepts introduced by SWC and provides a short overview of how the SWC client-server architecture could evolve in a near or more distant future.

Chapter 3 introduces web services in general, in the business world as well as in games.

Chapter 4 offers a brief overview of the technologies involved. It proposes a walk through the web services layers stack and describes various mechanisms to ensure web services security.

Chapter 5 defines the system specifications and limitations. It also describes the authentication and authorization mechanisms that have been designed for this specific application. Finally, it presents the various clients built by the author to demonstrate that the web services components are working as expected.
The last part of this thesis introduces the various prototypes created by external client developers. It shows how the web services were welcomed by the community, as illustrated by some statistics about use.
Chapter 2

Massively multi-player on-line role-playing games

2.1 What is a MMORPG?

A Massively Multi-player On-line Role-Playing Game, briefly MMORPG, is a multi-player computer role-playing game that enables thousands of players to play in an evolving virtual world at the same time over the Internet. Some of the most popular MMORPGs are EverQuest (Figure: 2.1(a)), the new World of Warcraft (Figure: 2.1(b)) and Dark Age of Camelot (Figure: 2.1(c)).

In role-playing, participants adopt characters that have personalities, motivations and backgrounds different from their own. Role-playing participants are actors in an improvisational or free-form drama or theater.

RPGs, or role-playing games, organize the possibility to play a character in fictional stories. Originally RPGs, such as Dungeons & Dragons, were supported by paper, pencil or other material elements. Later on, role playing games were made available as computer applications, e.g., the Final Fantasy series. The internet revolution introduced games such as Diablo, allowing multi-player interactions.
MMORPGs follow a client-server model. Players, running the clients, are represented in the game world by an avatar. Providers (usually the games’ publishers) are responsible for the server side and host the virtual worlds in which the characters evolve. Once a player enters the virtual world, he can engage into a variety of activities with other players. All these interactions within the virtual world, always available, and an ever-changing, world-wide stream of players animating individualized avatars characterize the Massively Multi-player On-line Role-Playing Game.

While in some MMORPGs, the virtual world surrounding the characters only evolves while the player is on-line, other MMORPGs use persistent worlds. A persistent world is a virtual world, which continues to operate even after the player has quit the game.

\(^1\)Graphical representation of the character
Characters in such games can even perform actions while the player is actually off-line. The game continues off-line: so even if all players were to leave the game at the same moment, or if the server supporting the game would suddenly be disconnected from the Internet for any reason, the world would keep running.

While, as in any other games, the developers are responsible for creating the program and solving the bugs, MMORPGs’ developers are also in charge of supervising the virtual world. They should constantly offer to the players updates, new activities and enhancements to sustain their interest.

Most MMORPGs are offered for sale on the market: players must either purchase the client software or pay a monthly fee in order to access the virtual world. Games of this type are immensely popular, with several commercial MMORPGs reporting over 200 000 subscribers. However, some totally free-of-charge games (e.g.: Daimonin Figure: 2.2) may be found on the Internet, although their quality is generally lower when compared to their “pay-to-play” counterparts.

### 2.2 Browser-based MMORPGs

With the success of the MMORPG genre in recent years, several multi-player games played began using web pages as game interface. Browser-based MMORPGs are usually simpler games than their not-website based counterparts.
Chapter 2. Massively Multi-Player On-Line Role-Playing Games

Browser-based MMORPGs typically imply turn-based play and simpler strategies, although many interesting variations on popular themes can be found: in Dark Galaxy, players control planets and fleets of ships; in Kings of Chaos, the player commands a whole army instead of a single character.

Kings of Chaos (Figure: 2.3) is an extremely popular browser-based MMORPG, with players numbering in the hundreds of thousands. Kings of Chaos’ popularity is primarily fueled by a reciprocal link clicking system, whereby each player can give another more soldiers by clicking on the latter’s unique link. This feature of the game has highly contributed to its promotion and its success.

Another example of a click based MMORPG is the Legend of the Green Dragon (Figure: 2.4), whose code is open source, allowing anyone to copy and modify the program. It can then be installed on anyone’s own server.

Figure 2.3: Kings of chaos race selection page

Figure 2.4: Legend of the Green Dragon
Kingdom of Loathing (Figure: 2.5), yet another browser-based MMORPG, presents a parody of the others: this feature has greatly contributed to its success.

Figure 2.5: KOL logo

Not all browser-based MMORPGs are turn-based text games. More recently, faster computers and Java have allowed the introduction of graphical browser-based MMORPGs, such as RuneScape (Figure: 2.6), which are more similar to standalone MMORPGs.

Figure 2.6: Runscape
2.3 Star Wars Combine overview

As already mentioned above, SWC is defined as follows by its developers:

“The Star Wars Combine is a free massively multi-player on-line role-playing simulation game, based on the Star Wars universe, developed by amateurs during their spare time.”

SWC is not only a MMORPG, it also offers the best Star Wars universe simulation fans may ever dream of. It has its own various game mechanisms related to the physics, economy, politics or engineering in the universe. Very challenging, it requires from the player both logical and socialization skills. All those mechanisms are defined by game rules that are carefully debated between players, written by the game administrator and made available to the whole SWC community.

As in any RPG, the character is one of the most important game concepts. Any player joining the game is invited to create his own character with various specific skills that he will be able to enhance throughout the game. In most MMORPGs when a player disconnects, his character is saved in the game database and his avatar disappears from the game. When the player returns, his character reappears somewhere in the game universe.

SWC has adopted a different approach than those MMORPGs. Once a player creates a character, his character remains in the game universe until it gets killed. This is referred as characters’ persistence.

Another main concept is the concept of factions. Factions are groups of players who are organizing and managing themselves. SWC proposes a complete system allowing any player to create his own faction whenever he wishes. He will, however, have to fulfill a few in-game conditions such as a determined capital, in-game cash, or in-game assets as well as out-of-game conditions such as the creation of a web site. Upon foundation of a new faction, the player may again decide in which area his faction will work and develop
CHAPTER 2. MASSIVELY MULTI-PLAYER ON-LINE ROLE-PLAYING GAMES

Figure 2.7: A SWC character

e.g. a faction may specialize in bounty hunting, mining, trading or any other activities. The player will also have define the ultimate objective of the faction, for example to rule the galaxy.

Figure 2.8: A SWC faction

In order to manage themselves, factions can use tools made available to them through the game interface. While those tools are very well designed and offer various ways to manage the faction, the possibilities to export the data remain limited. While exporting data does not seem very useful for small factions, it is a very attractive feature for most of the medium to large scale factions, that have most of the time developed a dynamic web site as well their own application and tools.

Today, SWC counts more than 2000 active players, all interacting with the game engine through a web-based game interface named Darkness. This graphical user interface, briefly GUI, is the only existing way to interact with the game engine. Although the GUI
is constantly being updated by the developers, it will always present some drawbacks, such as making machine to machine interactions difficult to develop.

The purpose of creating a web services infrastructure is to allow communications between heterogeneous machines and the game engine. Such an infrastructure would be very useful, because the faction concept has always been an essential key to the SWC success. In particular, in large factions, groups of players tend to develop their own tools to improve their management capacities and enlarge their game experience. Web services would allow those tools to be automatically updated with fresh data.

### 2.4 Evolution of the SWC client-server architecture

#### 2.4.1 Today: direct user interaction

The SWC game technology is an on-line browser-based game model. Players connect to the SWC server, using their web browser. The server runs an application to execute the game simulation (called Server2.0) and generate web pages to interface with the players. The whole game is centralized in one location and no information is transmitted to third parties’ web sites.

![Client-server model](image)

Figure 2.9: Client-server model

Most players also connect to their faction web site but no direct interaction exists between the SWC server and the faction web site. Update of users’ accounts, characters’ position, characters’ belongings and so on are not automatically done, which adds substantial workload for factions web-site administrators.
This lack of automatic updates complicates the game as the players have to register accounts on various web sites before logging in. They also have to get used to the different interfaces. These complications make cheating easier.

### 2.4.2 Tomorrow: direct trust relationship

In the direct trust relationship model, factions are able to directly connect to the SWC server to exchange informations. Informations can travel in both directions: factions update their databases with the galactic events occurred on the SWC server, while the SWC server database is being fed with faction decisions. Those decisions are taken by the faction on the basis of its own management interface and remotely executed through the use of web services.

![Figure 2.10: Direct trust model](image)

Darkness becomes just a graphical user interface and is strictly separated from the SWC server. This separation provides the following advantages:

- It offers the possibility to use different programming languages and therefore the most efficient for each purpose;

- Darkness and SWC server can be located on different physical machines, permitting load balancing; and
Darkness developers can concentrate their work on functions that are not part of the faction tools management scope.

However, as Darkness is still managed by the game server developers, it has kept privileged access to most remote functions available.

Single Sign-On login becomes available. It allows players to use a single account to connect to all the various web interfaces. This helps reducing the number of cheaters that are trying to abuse the factions (spies, multi-accounts, etc). Players can choose the interface they prefer, either Darkness or their faction web site depending of the tools needed. They can come and go from one to the other without having to log in several times. If both have the same “look and feel”, faction web sites and Darkness will soon or later end up looking as extensions of one another.

### 2.4.3 The day after tomorrow: indirect trust relationship

In an indirect trust relationship, factions offer web services to one another, using security mechanisms to authenticate and authorize access to their services.

In this situation, shown in Figure 2.11, players are able to use functionalities made accessible by the different factions belonging to a the SWC trust domain. Players can login to any faction web site and, assuming permission is granted, make use of any other faction web site to execute different tasks, for example make use of the web services of a trading faction for buying materials. The trading faction itself could make use of a transportation faction web services to organize the delivery and integrate this convenience as part of its products.

Darkness will still be running as any other classic client application that is able to connect with web services. It can be used by players that do not want to use other faction interfaces.

The diagram, shown in Figure 2.12 could become even more complex if the factions develop their own locally executable applications. Those will become parts of the SWC trust domain and will be able to interact with other actors such as the gamer server, the trading faction, the transportation faction, etc.
CHAPTER 2. MASSIVELY MULTI-PLAYER ON-LINE ROLE-PLAYING GAMES

New applications, created from scratch by factions, coded in various programming languages, such as Java or C#, will be running on the players’ computers. Such new applications could provide interesting new interfaces. Connected directly to the faction web sites, they would still be able to use the game server web services. They will offer to the players improved game experience. Faction applications may be built to meet players requirements.
From this point, the game can really evolve the way players want it. We can imagine that a client application would integrate a 3D interface allowing the player to move in a room. Factions could also set up shooting practices as part of the self defense class of their educational program.
Chapter 3

Web services

3.1 What are web services?

The W3C's Web Services Architecture Working Group has come to an agreement on the following working definition of a web service:

“A Web service is a software application identified by a URI, whose interfaces and bindings are capable of being defined, described, and discovered as XML artifacts. A Web service supports direct interactions with other software agents using XML-based messages exchanged via Internet-based protocols.”[7]

Web services are software components designed to support machine-to-machine interactions over a network. They have interfaces described in a machine-processable format. Any system can interact with the web services in the manner prescribed by their descriptions using messages, typically conveyed using HTTP[2] with an XML serialization in conjunction with other Web-related standards.

1The World Wide Web Consortium (W3C) develops interoperable technologies (specifications, guidelines, software, and tools) to lead the Web to its full potential. W3C primarily pursues its mission through the creation of web standards and guidelines. In its first ten years, W3C published more than eighty such W3C Recommendations. Currently, the W3C has 394 Members

2HyperText Transfer Protocol is the primary method used to convey information on the World Wide Web. The original purpose was to provide a way to publish and receive HTML pages.
3.2 Why to use web services?

Web services wide acceptance in the business world is explained by the many advantages they offer. The first purpose of web services is to procure a high level of integration.

Web services are self-described: neither the client nor the server knows or cares about anything besides the format and content of the request and response messages (loosely-coupled application integration). With UDDI and WSDL, the web service description and discovery can be automated.

Web services can be deployed in heterogeneous environments. The interaction between a service provider and a service requester is designed to be completely platform and language independent. Web services can be combined: simple web services can be aggregated to more complex ones, either by using workflow techniques or by calling lower-layer web services.

The advantages and disadvantages of web services are summarized below:

Advantages

• Web services provide interoperability between various software applications running on disparate platforms.

• Web services use open standards and protocols. Protocols and data formats are text-based, making it easy for developers to read and comprehend.

• By using HTTP, web services can work through many common firewall security measures, without requiring any changes to the firewall filtering rules.

• Web services allow software and services from different companies and locations to be easily combined to provide an integrated service.

• Web services allow the reuse of services and components within an infrastructure.

Disadvantages

• Web services standards for features, such as transactions, are currently nonexistent or still in their infancy compared to more mature distributed computing standards.
• Web services suffer from poor performance compared to other distributed computing approaches. This is a common trade-off when choosing XML or text-based formats.

• By utilizing HTTP, web services can pass through existing firewall security measures whose rules are intended to protect or audit communication.

3.3 Web services in the business world

Web services are not a technology just promoted by a few “web gurus” and are not just meant for laboratory experimentations. Web services have been conceived to respond to today’s inter-business technological issues. Their primary objective is to offer integration and interoperability to business actors regardless to their environment.

Sadly, many promising standards responding to business issues have never passed the stage of prototypes. Most of them were never integrated into production servers and never used by the industry world. However, this doesn’t seem to be the case with web services. They are being adopted by many major companies trying to improve their e-commerce products.

Here are a few examples:

• Google3 (Figure: 3.1(a)):
  Google created a web services API allowing clients to query more than 8 billion web pages directly from their own computer programs and to receive results as structured data. Furthermore, clients can access information from the Google cache and check the spelling of words.

• eBay5:
  eBay, the famous online auction company, permits its web services users to list items for sale, to search in the product database, to retrieve and give feedback, to get the list of bidders for a given item, to list items being sold by a given seller, etc.


4. Application Program Interface: set of methods prescribed by a computer operating system or by another application program by which a programmer writing an application program can make requests of the operating system or another application.

There are more than 70 web services available today and even more under active development.

Google Web APIs (beta)

Develop Your Own Applications Using Google

With the Google Web APIs service, software developers can query more than 6 billion web pages directly from their own computer programs. Google uses the SOAP and WSDL standards so a developer can program in his favorite environment—such as Java, Perl, or Visual Studio .NET.

To start writing programs using Google Web APIs:

(a) Google web services API

Getting Started

This document is intended to be the first information you need about Amazon E-Commerce Service (ECS). This guide will help you register to use ECS and help you find the resources you need to get your ECS-powered application underway.

Step 1: Registering as an Amazon Web Services Developer

In order to access ECS, you must first register with the Amazon Web Services program. Registration is free and takes only a few minutes. Once your registration is completed, you will be assigned a subscription ID that will allow you to access all ECS functionality.

Register on the Amazon Web site

Step 2: Evaluation and Planning

ECS has unlimited potential for a variety of applications. You're only limited by your own imagination and creativity. You can integrate ECS with your own web site or application in a wide variety of ways.

- Take a tour of the features and see ECS in action for yourself.
- Read the ECS Application and Planning

(b) Amazon web services API

Figure 3.1: Web services in the industry

- Amazon[6](Figure: 3.1(b)):
  Amazon offers access to its catalog via web services. Customers can now integrate Amazon vast on-line content within their own application. Web services such as searching the catalog, looking up data for specific products, getting seller feedback for non-Amazon vendors and looking up customers' wish lists are available.

- Paypal[7]
  PayPal is an Internet bank, which allows the transfer of money between email users, avoiding traditional paper methods such as checks or money orders. PayPal performs payment processing for e-commerce vendors, auction sites, and other corporate users. Performing a payment, obtaining transaction details, searching for a transaction are now actions that can be performed through web services.

- ViaMichelin[8]
  ViaMichelin, a online guide that provides route planner and interactive maps to pre-

pare personalized itinerary, has added web services as part of their products. Clients can significantly enhance their application by adding geographical information and door to door route calculations.

All these companies are trying to increase their number of clients not only by adding or changing their products but by increasing their market targets. They are transforming their web site application into a platform to allow other applications to integrate the various functionalities they offer.

“A platform is something that other people build on top of. You know you have an effective platform when someone builds something you never predicted.”

—Pierre Omidyar

eBay Founder and Chairman

To facilitate this migration, major actors in the computer industry have started to include web services tools into their Integrated Development Environment \(^9\) briefly IDE. These toolkits make web services very easy to code for the programmers. Among those IDEs, one can cite: Sun Netbeans 4.1, Microsoft WSE 2, IBM WebSphere and the open source platform Eclipse 3.1.

3.4 Web services in games

To the best of our knowledge, web services have not yet been implemented in MMORPGs. The only existing games that make use of web services are very basic ones, such as Tic Tac Toe or Chess. But these games are not large game, involving sophisticated game mechanisms. Instead they are used as experimentation or web services tutorials, unlikely to be ever considered as true games.

The only game apparently using a web service like mechanism is EverQuest2 made by Sony. In this game, players form guilds that, as in SWC, are required to build their own web site. Guilds receive a web space on one of the game servers. On these pages, they can dynamically access data that are available in the virtual world. Sadly, while the concept

\(^9\)An integrated development environment is computer software to help computer programmers develop software.
seems interesting, it does not offer data exchange with other web site hosts and it’s hard to believe that it implements any of the web services standards.

YaYa LLC develops Internet games used in marketing programs. While consumers play YaYa games and try to collect points for prizes, they also provide data for corporate Customer Relationship Management systems (CRMs). While filling databases is a very difficult task in most CRM systems because it’s difficult to get customers to reply to surveys, Yaya offers a promising solution.

YaYa has developed a Consumer Dialog Technology (CDT) platform using web services capabilities that allow software designers to create games capturing data on a player’s
preferences e.g. for car models. This information is then delivered to the CRM systems, e.g. the car manufacturer.

Yaya plans a next generation of advertising technology with many more sophisticated role-playing games, through which the auto dealers would get extended information on which vehicles each player is interested in.
Chapter 4

Technological overview

4.1 The web services layers and interoperability stack

4.1.1 Layers

Web services are composed of various layers that together form the web services stack. XML, the Extensible Markup Language, is a standard supported and accepted by thousands of vendors worldwide. XML lies at the very base of the stack. It offers interoperability, one of the first objectives of web services. While XML provides the syntax for messages, HTTP is the underlying protocol, allowing for the exchange of messages required for web services communications. But HTTP is not required, other transport protocols can also be used. Furthermore, Web services don’t need to be developed on the World Wide Web. A web service can live anywhere on the network, Inter- or intranet.
When most developers think of web services, they think of SOAP\(^1\) which is the “accessed” part of the web services definition. But, as shown in Figure 4.1, SOAP is only the XML-based message protocol used to communicate with Web services. SOAP is the underlying plumbing for Web services because it is the protocol that everyone agrees with.

But, as we’ll see later, Web services use many other technologies than SOAP. UDDI\(^2\) or ebXML registries allow applications to dynamically discover informations about Web services. Once discovered, a Web service need to be described. The message syntax for a Web service is described in WSDL, the Web Service Definition Language.

WSFL, the web service flow language, is used to describe how web services can be aggregated and combined together.

![Figure 4.2: Web services scenario](image)

### 4.1.2 The transport layer

At the basis of the stack is the transport layer. Web services are basically messaging mechanisms, so it’s reasonable to think about message transport technologies as the basis for any web services conceptualization.

Web services are essentially transport neutral: a Web service message can be transported using the ubiquitous HTTP or HTTPS protocol, as well as any other transport

---

\(^1\)Simple Object Access Protocol

\(^2\)Universal Description, Discovery and Integration
mechanism such as SMTP. I won’t go into any further details about the transport protocols, as web services do not request their developers to know all the specificities of the message transport layer.

4.1.3 The messaging layer

XML is a general-purpose markup language recommended by the W3C for creating special-purpose markup languages. It defines a syntax used to format data, without using any predefined tags. XML allow developers to define by themselves the elements they will need and when they will need them. The high degree of flexibility provided by XML explains its wide acceptance in various domains.

XML provides a grammar to format the tags used in the document. The grammar defines the position and the order of the tags, the names of the elements being used, the way attributes are to be linked to the elements and so on... The grammar allows the creation of XML parsers\(^3\) that are able to read and interpret XML formatted documents. Documents conforming to all of XML’s syntax rules are known as “well-formed”.

```xml
<person sex="female">
  <firstname>Anna</firstname>
  <lastname>Smith</lastname>
</person>
```

An XML scheme is a grammatical description of a type of XML document. It expresses the constraints related to the structure and content of that type of document. These constraints come on top of the basic constraints imposed by XML itself. A number of standard and proprietary XML scheme languages have emerged for the purpose of formally expressing such scheme, and some of these languages are XML-based.

An XML document that complies with a particular scheme, in addition to being well-formed, is said to be valid. It conforms to a particular set of user-defined content rules that describe correct data values and locations. Documents need to be both well-formed and valid to be correct.

\(^3\)A parser is a computer program that carries out the task of analyzing a continuous stream of input in order to determine its grammatical structure with respect to a given formal grammar.
4.1.4 Accessing web services with SOAP

SOAP is a remote procedure call mechanism like RMI or CORBA. It is defined has a lightweight protocol, allowing exchange of structured information between entities. SOAP was conceived to be used in massively distributed environments such as the Internet. SOAP respects the layering model and is not making any assumption about the underlaying protocols on the top of which it is staying. Although HTTP is certainly the most widely used transport protocol, one could imagine using SMTP or any other bidirectional communication protocol.

SOAP is of great importance because it represents the industry’s best effort to date to standardize the infrastructure technology for cross-platform XML distributed computing. At its heart, SOAP is a specification for a simple, yet flexible XML application. SOAP provides for the following:

- A mechanism for defining the unit of communication:
  All information is packaged in a clearly identifiable SOAP message. This is done via a SOAP envelope that encloses all other information.

- A processing model:
  This defines a well-known set of rules for dealing with SOAP messages in software.

- A mechanism for error handling:
  Using SOAP faults, one can identify the source and cause of an error and it allows error diagnostic information to be exchanged between participants.

- An extensibility model:
  SOAP headers make it really easy to implement arbitrary extensions on top of SOAP. Headers may contain additional pieces of information, which will travel along with the main message. These special pieces of informations may be intended for different nodes along the message path.

- A flexible mechanism for data representation:
  This mechanism allows for information already serialized in some other format, such as text, XML or representation of programming languages data types, to be included in the message.

- A protocol binding framework:
  The framework defines an architecture for building interfaces to send and receive SOAP messages over arbitrary underlying transports.
CHAPTER 4. TECHNOLOGICAL OVERVIEW

Let’s now take a quick look at what kind of elements a SOAP message contains:

- A SOAP envelope that wraps the message
- A description of how data are encoded
- A SOAP body that contains the application-specific message that the back-end application will understand.

![Figure 4.3: Structure of a SOAP message](image)

SOAP offers the following advantages:

- Simplicity:
  SOAP fully relies on existing and well-known technologies. Its deployment only requires a XML parser, a HTTP server and few lines of code. Furthermore, most of the time libraries are available and often integrated into the IDE. SOAP deployment becomes then even easier than RMI, and a developer can quickly terminate the learning phase and start coding the application.

- Portability:
  Because SOAP is an XML application and does not rely on any transport protocol or platform, it is very flexible and will be able to adapt to future technological advances. It offers a complete compatibility between languages and platforms.

- Easy Deployment:
  Used on top of HTTP for example, SOAP does not require any particular network configuration, unlike CORBA or RMI that require some ports to be open.
Readability:

SOAP benefits from XML readability; the messages exchanged between the entities are legible by humans.

Web services can be completely independent from the presentation layer or the graphical user interface of applications. Instead, Web services send data in XML format, and applications can add style and formatting when they receive data.

Here is an example of a SOAP message:

```xml
<SOAP-ENV:Envelope
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/
    xmlns:si="http://soapinterop.org/xsd"
    xmlns:tns="https://ws.swcombine.com/swcws.php">
    <SOAP-ENV:Body>
        <ns1:WSgetCgtObjectResponse xmlns:ns1="urn:swcws">
            <return xsi:type="tns:cgt">
                <years xsi:type="xsd:int">6</years>
                <days xsi:type="xsd:int">230</days>
                <hours xsi:type="xsd:int">1</hours>
                <mins xsi:type="xsd:int">1</mins>
                <secs xsi:type="xsd:int">21</secs>
            </return>
        </ns1:WSgetCgtObjectResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Web services provide no graphics, they only forward XML messages to the client application. Because XML data is sent back and forth, any front-end application that understands XML can speak to the web service. Web services do not deal with the presentation layers, their main focus is the business logic. This feature makes them easier to reuse as software components.

Separating business logic from the presentation layer is commonly known in software engineering\[19\] as the Model View Controller, briefly MVC, paradigm. Web services abide by this paradigm. The user interface details and the business logic are separated in two different components while the layer between them, the controller, organizes the communication.
Because the presentation layer is separate from the web services, the information can be easily presented in many different ways. This is an interesting aspect because many browsers facilitate the offloading of the presentation process with style sheets.

### 4.1.5 Describing web services with WSDL

Whereas SOAP is the communication language for accessing web services, WSDL describes both the request and response messages that are always present in any web service transactions.

![WSDL structure](image)

WSDL, like SOAP, is an XML grammar. The W3C defines WSDL as “an XML format for describing network services as a set of endpoints operating on messages containing

---

4 Style sheets describe how documents are presented on screens, in print, or even how they are pronounced. [32]
either document-oriented or procedure-oriented information.” To know how to send messages to a particular web service, an application will parse the WSDL documents and dynamically construct the appropriate SOAP messages.

![Diagram of Dynamic communication with WSDL]

**Figure 4.6: Dynamic communication with WSDL**

WSDL describes the operational information (also known as the function description):

- **What the service does:**
  The operations the service provides, and the data needed to invoke them

- **How to invoke the service:**
  Details of the data formats and protocols necessary to access the service’s operations

- **Where the service is located:**
  Details of the protocol-specific network address such as an URL

Most developers do not have to understand WSDL and SOAP details to create web services, as most toolkits create the SOAP message parsing and the WSDL automatically.

### 4.1.6 Finding web services with UDDI

UDDI (Universal Description, Discovery and Integration) is a platform-independent framework for storing web services descriptions, discovering web services provider businesses, and integrating web services by using the Internet. It is a cross-industry effort driven by major platforms and software providers.
like Dell, Fujitsu, HP, Hitachi, IBM, Intel, Microsoft, Oracle, SAP and Sun, as well as a large community of marketplace operators and e-business leaders.

UDDI is a directory for storing business related information about web services and their interfaces described by WSDL documents.

The information provided in a UDDI registration consists of three components: white pages, the yellow pages and the green pages.

The white pages are the equivalent of business cards catalog. They include basic business information, such as a description of the company in different languages, points of contacts with email addresses and phone numbers, and links to external documents that go into further details.

The yellow pages categorize the services provided according to standard taxonomies.
Finally, the green pages document the technical information related to the service available e.g. how to invoke the web services with the use of WSDL documents, what are the authorization rules etc.

![](image)

**Figure 4.9: UDDI role**

UDDI has two functions:

- allowing businesses to register their web services
- allowing other organizations to “browse” the information available to look for businesses and the services they provide.

Any business can decide to register itself. Once the information has been registered, other organizations can access it. Obviously, there are security concerns when placing information on a public registry. Like for an Internet web site, the decision as to which information should be made public in an UDDI registry needs to be carefully analyzed.

### 4.1.7 Orchestrating web services with WSFL

Orchestration is the process of combining simple web services to create complex, sequence-driven tasks. This process, sometimes called “flow composition”, involves creating business logic to maintain conversations between multiple web services. Orchestration can occur between an application and multiple web services; or multiple web services can be chained into a work flow, so that they can communicate with one another.
CHAPTER 4. TECHNOLOGICAL OVERVIEW

Figure 4.10: Recursive composition of web services

The Web Services Flow Language, known as WSFL, is “an XML language for the description of Web Services compositions”[21]. WSFL identifies two types of Web Services compositions. The first type (flow models) specifies the appropriate usage pattern of a collection of Web Services, in such a way that the resulting composition describes how to achieve a particular business goal; typically, the result is a description of a business process. The second type (global models) specifies the interaction pattern within a collection of web services; in this case, the result is a description of all web services interactions.

4.2 Securing web services

4.2.1 Security requirements

There is always a security risk when exchanging information over insecure networks such as the Internet. Five security concerns must be addressed to ensure the safety of information exchange:

- Confidentiality guarantees that exchanged information is protected against eavesdroppers.
- Integrity refers to the assurance that a message isn’t modified during transit.
- Non-repudiation guarantees that the message sender can’t deny having sent it.
- Authentication guarantees that access to remote applications and data is restricted to those who can prove their identity.
• Authorization is a process to verify whether an entity with a given identity can access a particular resource.

4.2.2 SSL

Secure Socket Layer protocol, briefly SSL, is a point-to-point protocol that can be used for mutual or one-way authentication and it is used to encrypt data between two points. In environments with a single client and server, a HTTPS session may be enough to protect the confidentiality of the data transmitted.

![Diagram showing protection at every point](image)

Figure 4.11: Protection at every point

However, in a multiple-points scenario, where a client user connects to a portal, which itself is connected to a web service, itself connected to yet another one, one or more SSL connections will not pass on the proof of the original user’s authentication and authorization credentials between all those nodes: the assurance of the integrity of the message will get lost all the more that there are nodes between the original client and the ultimate web service.

4.2.3 XML signature

XML Signature is a W3C Recommendation that provides a means to validate the integrity and non repudiation of a message circulating between multiple nodes. With the XML Signature, any part of an XML document can be digitally signed. In fact, multiple parts of an XML document can be signed by different people or applications.
XML Signature relies on public key technology in which the hash of a message is cryptographically signed. Because of the nature of public key signatures, anyone with the signer’s public digital certificate can validate that the signer indeed signed the message. This procedure provides legal proof that the signer cannot refute. It guarantees the message’s origin and content and can also serve to authenticate SOAP messages.

4.2.4 XML encryption

XML Encryption is also a W3C Recommendation that handles confidentiality for messages circulating between multiple nodes. It can hide sensitive content, so that only the intended recipient can read the sensitive information. In an XML file, different parts of the document can be encrypted, while other parts can remain unencrypted. This can be helpful with web services, when messages may circulate between multiple points before the receiver gets the message.

Different encryption mechanisms can be used, including symmetric and public key encryption. If confidentiality is requested for some web services, a part of the application-
specific SOAP message may be encrypted. Although the message may travel through many servers, only the intended recipient should be able to read the message.

4.2.5 SAML

The Security Assertions Markup Language, or SAML, is being developed by the OASIS security services technical committee. SAML is an XML-based framework for exchanging security information.

As a framework, SAML deals with three things. First, it defines syntax and semantics of XML-encoded assertion messages. Second, it defines request and response protocols between parties seeking security information. Third, it defines rules for using assertions with standard transport and message frameworks. For example, it defines how SAML assertion messages can be transported using SOAP over HTTP.

The security information exchanged is expressed in the form of assertions about subjects, where a subject is an entity (either human or computer). Each entity carries an identity in some security domains. Assertions can convey information about authentication acts performed by the subjects, attributes of the subjects, and authorization decisions.

---

[^5]: Organization for the Advancement of Structured Information
concerning the access to certain resources by the subject. Assertions are represented as XML constructs and have a nested structure.

One major design goal for SAML is Single Sign-On, also known as SSO. It provides to a user the ability to authenticate in one domain and use resources in other domains without re-authenticating. However, SAML can be used in various configurations to support additional scenarios as well. Several profiles of SAML are currently being defined that support different styles of SSO and the securing of SOAP payloads.
4.2.6 XACML

The eXtensible Access Control Markup Language, or XACML[24], is an OASIS standard that allows administrators to define the access control requirements for their application resources. The language support includes data types, functions, and logic which allow complex (or simple) rules to be defined. XACML also includes an access decision language used to represent the pending requests for a resource. When a policy, which protects a resource, is found, functions compare attributes in the request against attributes contained in the policy rules, ultimately granting or denying access.

Figure 4.16: XACML model

When a client makes a resource request upon a server, the entity charged with the access control is called the Policy Enforcement Point. In order to enforce the policy, this entity will formalize attributes describing the requester at the Policy Information Point and delegate the authorization decision to the Policy Decision Point. Applicable policies are located in a policy store and evaluated at the Policy Decision Point, which then returns the authorization decision. Using this information, the Policy Enforcement Point can deliver the appropriate response to the client.

4.2.7 WS-Security

The WS-Security specification defines a format to include security tokens and mechanisms to protect SOAP messages. Digital signature serve as integrity and non repudiation checks to ensure message protection, and encryption guarantees confidentiality. In addition, WS-Security provides a flexible mechanism to include various claims in SOAP
messages with security tokens. With message protection security tokens, WS-Security can provide a basis for other specifications.

```xml
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP:Header>
      <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
        ...
      </Signature>
      <EncryptedKey xmlns="http://www.w3.org/2000/09/enc-enc-enc#">
        ...
      </EncryptedKey>
      <wsse:UsernameToken xmlns="http://schemas.xmlsoap.org/ws/2003/06/secext">
        ...
      </wsse:UsernameToken>
    </wsse:Security>
  </SOAP:Header>
  <SOAP:Body>
    ...message body...
  </SOAP:Body>
</SOAP:Envelope>
```

### 4.2.8 Other web services security specifications

Web services address a complicated problem: loosely coupled, dynamically configured heterogeneous distributed computing. However, the technology is doing this in a clever way: instead of building Web services as a single, monolithic specification, the community has chosen to incrementally deliver a series of smaller purpose-focused specifications. Each specification deals with its own domain.

Many security specifications are built on top of WS-Security. While most of them are not yet finalized, a diagram of the specification release is summarized in Figure 4.17.

![Figure 4.17: Roadmap of web services security specifications](image)

The specifications shown in Figure 4.17 can be summarized as follows:
• WS-Policy provides a framework for describing web services meta-information.

• WS-Trust prescribes an interaction protocol to access Security Token Services.

• WS-SecureConversation defines a security context with which parties can share a secret key to sign and encrypt parts of messages efficiently.

• WS-Federation provides a framework for federating multiple security domains.

• WS-Privacy provides a framework for describing the privacy policies.

• WS-Authorization defines how to exchange authorization information among parties. The authorization is defined as a security token.

The job of a web services designer is to determine which specifications his system needs and to implement them accordingly. The question of how to choose a design for a SWC web services infrastructure will be dealt in the next chapter.
Chapter 5

Solution design

5.1 Specifications

The goal of this thesis is to implement a web services infrastructure within the current SWC production server to demonstrate the usability of web services mechanisms.

On the server-side, the web service infrastructure has:

- to be easy to modify and upgrade:
  The code has to remain simple, structured and well documented to insure later evolutions;

- to maintain server and game-engine integrity:
  This includes avoiding both server attacks and character cheats;

- to provide appropriate web services management tools:
  These should allow for the various aspects of the web services server to be handled by administrators with no or little technical knowledge. Furthermore the administrators should not be burdened with practical implementation details;

- to be isolated and independent from the rest of the server code:
  SWC involves many programmers working on various features. As the web services component is new, it can easily be made independent from the rest of the code; strict separation will not cause any trouble. To the contrary, using various layers will be advantageous as it will allow independent evolution of the libraries and the web services using them;
• to prove its utility through user feedback:

Community acceptation is a must, it’s the last but also the most important point to insure that the web services approach will remain viable.

Additionally to this infrastructure, we’ll be creating web services prototypes on the client side. The creation of a client prototype is needed for three reasons:

• to test the server-side architecture;
• to motivate developers to create their own client;
• to obtain feedback and check the architecture usability;

Client creation require from faction developers a lot of their spare time as most of them are unfamiliar with web services. Particular attention has to be paid when interacting with them. This means that communication skills are needed when giving explanation, describing known bugs, integrating new features or updating the documentation. Good communication gives to developers the impression that the project is very dynamic and it motivates them a lot. This psychological aspect, which is often forgotten about, is very important, in particular in a community of free developers such as the SWC.

5.2 Web services motivation

The decision of using web services technologies to enable remote faction management was made for the following reasons:

• Faction communication has clearly a distributed computing nature: parts of the solution exist in multiple network endpoints. This is particularly true if further developments have to reach the indirect trust relationship model described in Chapter 2;

• The solution has to be built and run by various organizations involving different administration teams. The multiplicity of partners calls for a dynamic approach, which XML protocols associated with web service technologies will favor;

• The various components need to be integrated on heterogeneous platforms (operating system, application server or programming language);
• Most of the transactions should become automated and therefore extensive use of machine-to-machine communication is preferable;

• The solution needs to be flexible to respond to the various changes that future developments could require from any partner;

• The data need to be available to more than just the core application that generates and maintains them. Web services are a simple approach for making information accessible to all sorts of different clients.

5.3 Technical constraints

5.3.1 Server environment

PHP

PHP, an open-source programming language, is widely-used primarily for server-side applications and developing dynamic web content. PHP’s ease of use and similarity with the most common structured programming languages (like C and Perl) allows most experienced programmers to start developing complex applications with a minimal learning curve. It also enables experienced developers to get involved with dynamic web applications without having to learn a whole new set of functions and practices.

Up until version 3, PHP had no object-oriented (OO) features. In version 3, basic object functionalities were added. The same semantics were implemented in PHP 4 as well as pass-by-reference and return-by-reference for objects but the implementation still lacked the powerful and useful features of other object-oriented languages like C++ and Java. In version 5, PHP’s object-oriented functionalities have been very much enhanced and are more robust and complete.

SWC is, for historical reasons, written in PHP4, which lacks most of OO programming features. However, procedural programming offers the advantage to be easier and therefore more accessible to new programmers than an OO architecture. A procedural approach is preferred for web services development, as it could be fully integrated in the current game engine without disturbing programmers.

1 recursive acronym for “PHP: Hypertext Preprocessor”
MySQL

The server runs MySQL 4.0 to store the state of the game universe. MySQL 4.0 is a relational database. It has been carefully designed over the years in respect of the game relational model. MySQL offers the great advantage of being strongly integrated in PHP. But the database manager still lacks some useful features such as triggers and some parts of the SQL language.

While running a powerful database, the server’s simulation is using a lot of resources. Bottlenecks often occur at the database connection level. The web service infrastructure should not by any mean cause a major increase in the number of database requests.

5.3.2 Clients

Clients are applications developed by the faction programmers. While no assumption should be made about the language or environment in which the clients will be developed, it’s very likely that a vast majority will be using PHP4 and PHP5 for coding a dynamic web portal. Also, since most factions do not own a server but rather have their web site hosted by third parties, a particular attention as to paid to permission issues. Indeed, a lot of factions don’t have permission to install or upgrade any software on the server hosting their web site.

Other technologies such as C# or Java are also most likely to be used to run both dynamic web portals and locally runnable applications.

5.4 Libraries

5.4.1 Choosing the library

PHP does not come with a bundled SOAP extension. However a few toolkits are available that allow an easy integration of SOAP into applications. There are two major SOAP implementations for PHP: NuSOAP and PHP SOAP extension. The first has more features, but is written in 100% PHP, while the PHP SOAP extension is written is C.

NuSOAP is a group of PHP classes that allows developers to create and consume SOAP web services. It does not require any special PHP extensions. NuSOAP is a very dynamic
project. The last released version of NuSOAP is 0.9. It supports much of the SOAP 1.1 specifications. It can generate WSDL 1.1 and also consumes it for use in serialization. However, it must be noted that NuSOAP does not provide as complete a coverage of the SOAP 1.2 and WSDL 1.1 as some other implementations, such as .NET and Apache Axis, do.

![NuSOAP Graphical User Interface](image)

- NuSOAP is object oriented;
- NuSOAP is coded in PHP script and therefore is slow;
- NuSOAP supports automated WSDL generation which make it really easy to write the web service server-side component;
- NuSOAP handles SOAP arrays and complex data types;
- NuSOAP offers SSL encapsulation of SOAP messages but it requires the PHP's CURLE extension or Openssl to be installed.

PHP SOAP extension is written in C and not in PHP. This gives to this extension a huge speed advantage. The extension is currently marked as experimental, but should gradually become more stable and reliable over time. The SOAP extension implements a large subset of SOAP 1.1, SOAP 1.2 and WSDL 1.1 specifications. WSDL is used whenever possible in order to make the implementation of web services more straightforward.
CHAPTER 5. SOLUTION DESIGN

Our preference for developing the server-side went to NuSOAP, because it does not require any compilation or complex installation steps and can run under PHP4, whereas the PHP-SOAP extension runs only under PHP5. NuSOAP is also the only toolkit offering on-the-fly auto generated WSDL documents for the web services it offers.

5.4.2 Toolkit

This section provides a quick tour on how NuSOAP works and what mechanisms are used behind the scene when creating or invoking a web method.

SOAP does not specify a protocol for the transport layer but HTTP is the most commonly used. As such, the messages to the server are SOAP-XML requests wrapped in HTTP requests. Likewise, the responses from the server are HTTP responses that enclose SOAP-XML responses. On the client-side, developers do not want to have to worry about all the details of SOAP serialization and HTTP encoding. The SOAP package is taking care of these tasks.

The SOAP package consists of a library linked to the client code. Web services are invoked simply by calling the appropriate method in the SOAP package (typically specifying the service URL, service name and all required parameters). The first job of the SOAP package is to serialize this service invocation into a SOAP request. It then needs to encode that message in a HTTP request and send it to the specified URL.
When the SOAP server sends back a response, the SOAP package will do the reverse of what was described in the previous paragraph: it will decode the HTTP message and extract the SOAP message, deserialize the SOAP message and obtain the return value of the method call. The return value found is then passed to the original method called on the client-side.

On the server-side the mechanism is slightly more complex, as a listener process is needed. It also requires an implementation of the service itself. But aside from that, the SOAP package is used in a similar way as on the client-side.

The listener process is often implemented using a service running as a web application. The application server will be set up to pass all requests for a certain URL (the URL for the SOAP service) to a SOAP server application. The job of the listener is to extract the XML-SOAP message from the HTTP request, to deserialize it (thereby separating out the method name and the supplied parameters), and to invoke the service method accordingly. The result of the method is then serialized, HTTP-encoded and sent back to the client.
5.5 SWC design evolution

We took a look on the various technologies available, defined the system specifications and limitations. Let’s now have a look on how SWC has been designed and how web services integration will be done.

5.5.1 SWC current design

“Developing an ambitious project like a massive multiplayer online game (MMOG) with a band of merry cowboys is a challenge at the limit of sanity.”

–Jehan (Veynom) Snyers d’Attenhoven, Star Wars Combine Sim Master

The architecture of SWC is based on the client-server architecture. Any player interaction with the game simulation is done by sending packets to the server. When two players are communicating with one another, the information exchange is handled via the server. While the client-server technologies used by SWC have evolved a lot during the last years, the actual game is browser-based. The server is hosting 3 components: the database, the libraries and the Graphical Client Interface.
The database part is responsible for holding the state of the simulation, including the character profiles, the objects’ position, faction assets, and so on...

The libraries are responsible for insuring data integrity of the database, implementation of the game rules and execution of the simulation. Together the database and the libraries form what is referred as Server2.0. It fully holds all the simulation logic and data.

The GUI is a web interface. It is the only client that communicates with Server2.0. The HTML code is generated using PHP scripts using data returned by the various library features. Both the client (GUI) and Server2.0 are still located on the same computer. The sum of Server2.0 and the web based GUI is referred as Darkness and constitutes the game server application.

### 5.5.2 SWC web services design

Web services have 5 components:

- Remote objects and functions interface:
CHAPTER 5. SOLUTION DESIGN

Figure 5.6: Web services server components

Web services require the remote functions and objects to be declared. WSDL definitions and web services registration are grouped together in this component:

- Remote objects and remote functions endpoints:
The endpoint is the actual page from which the web services can be called. Endpoints should be simple files containing web service configuration lines used to load the web services declaration files. Also part of the interface component is the SOAP toolkit (nuSOAP libraries);

- Web service core logic:
It contains the authentication and authorization layers but also all the additional code required to match the current game rules. The web services core logic is linked to the SWC libraries to perform remote action invoked by users.

- Web services database:
It is used to hold the various states that are required by the web services core logic.

- Web services graphical interface:
It is composed of an automated documentation tool, a permission management tool and a web services statistics tool.

All these components are part of the new web services architecture. Let’s now discuss how the web services will be integrated into the actual production server.

Web services represent an alternative to the classical web interface used to interact with Server2.0. It is obvious that from a role point of view the web services will not replace
Darkness web interface; from an access point of view, web services will lay between the client and the libraries, next to the web interface.

A new set of rules, proper to web services, are created to handle web services access. These authorization rules are completely independent from the “In-Game” rules that belong to the simulation.

### 5.6 Security design overview

The communication mechanism shown in Figure 5.8 illustrates the various parts of the code that are sequentially called during a Web Service invocation on the server side.

Although we have already discussed the fact that SSL is only a point-to-point security mechanism, it is useful for securing direct client-server communication. This ensures server authentication and avoids clear XML messages to be handled over the Internet. However, such technology might require some configurations on the client-side. To avoid rejecting clients because of technical issues, non SSL HTTP access remains available.

When a request comes in, the raw HTTP message is handled by the listener process. Once the message has been parsed, the authentication and authorization modules enter into effect. Authentication and authorization steps are handled by the WSPolicy method. This method takes a few arguments such as a security token (referred to as sToken) and
the kind of access category the web service falls into. Further explanations about the web services categories will be given in the authorization section.

The authentication method is responsible for verifying that the sToken is valid and to retrieve the client and the user information. Once the authentication process has succeeded, all the authentication related data are saved into a new structure named sData\(^2\) and returned back the WSpolicy method.

---

\(^2\)Security Data
The authorization module uses the sData created during the authentication process to determine whether or not access to the web services shall be granted. In any case, the authorization process hands back a response to the WSpolicy method, that takes care of sending a SOAP error message, if any of the authentication or authorization steps have failed.

The next two sections will show in further details how the authentication and authorization processes are working.

5.7 Authentication

Some web services require special privileges to be used. The first step to achieve web services security is to authenticate who is trying to access a web service.

Client programmers are developing their own application independently from the server developers or the game administrators. Therefore, attention has to be paid to user authentication issues. No authentication mechanism allowing the client to handle sensitive data such as user names and passwords should be used, since it would open the door to abuses.

To avoid such situations, an authentication mechanism based on SAML authentication procedure[26] was implemented and slightly modified to meet the game requirements. This authentication mechanism allows a user to connect through a web portal using his SWC user name and password to access some of the game features.

The player is assumed to use a recent browser and to trust his/her faction to perform actions as requested.

The authentication procedure introduces the following entities:

- The player using a web browser;
- The faction web site running under PHP and trying to access web services functions;
- The SWC server that provides a authentication web page, web services, a database and the game engine.
5.7.1 Connecting to a public web service

Before getting more into the authentication details, let’s have a look at how a web service not requiring any authentication works.

A few definitions:

- User browser: player program that displays HTML documents;
- Faction web server: PHP web site that serves HTML documents and accesses to web services;
- Game engine: core of SWC game and where the rules are enforced;
- SWCWS: SWC Web Services, an interface for remote program to connect to.

![Diagram showing the connection to a public web service]

Figure 5.10: Connection to a public web service

The following steps are involved when invoking a public web service:

1. The user goes to the faction web site;
2. The faction web site misses some information and sends a request to the SWCWS;
3. The response comes and is treated by the faction web site;

4. The faction web site generates HTML for the browser to display.

### 5.7.2 Authentication with web services

**Some more definitions**

- User password: password of the user to authenticate on the SWC web site;
- Client password: password given to a client for authentication;
- PHP session ID: identification number to allow session creation under PHP;
- Contract ID: identification number of the contract. A contract is the proof of a mandate given by the user to a client to perform actions on his account;
- sToken: a security Token is provided by the authentication service to allow further processing with authorization.

**Procedure**

1. The user goes to the faction web site and clicks to login;

2. The faction web site redirects the user to the SWC login page giving its Faction ID number (A well known identification number identifying the faction);

3. The user completes the SWC login page (through secure HTTPS connection) by giving its user name and user password and is aware that by doing this he/she is allowing the faction to act in the game;

4. The SWC login page checks that the user name and user password match and creates a contract containing:
   - the user ID;
   - the client ID;
   - the contract number.

The contract is kept in the contract database on the SWC server. The SWC login page creates a session for the user allowing future contract creation to bypass further authentication needs;
5. The SWC login page redirects the user to the faction web page and provides a contract identification number to the faction;

6. The faction web site needs now to obtain a sToken in order to bypass the authorization server that protects restricted web services. Requests for a sToken are made by calling the WSgetsToken web service giving as parameters:

   - its faction ID;
   - its faction password;
   - the contract ID.

The WSgetsToken web service checks with the contract database if the contract is valid between the user and the faction. Checking is done by comparing contract ID, user name and faction ID. The IP of the client is recorded;
7. The faction web site can call a web service by providing the sToken received previously;

8. Future calls to web service can be made by providing the sToken;

9. The faction web site sends back information gathered in a nice looking page to the user. The sToken may be saved in session for later use.

### 5.7.3 Reusing a security token

![Diagram of Reusing a security token]

Figure 5.12: Reusing a security token

1. The user connects to the faction web page and provides the PHP Session ID;

2. The faction web site unserializes the sToken previously saved. And sends a request to a web service providing the sToken;

3. The faction web site receives the response and sends back a good looking HTML page to the user.
5.7.4 Logging out

Figure 5.13: Logout procedure

1. The user clicks on a logout link on the faction web site;

2. The faction web site calls the WSlogout web service. The contract is deleted from the contract database;

3. The faction receives an acknowledgment that the contract has been canceled and deletes the corresponding sToken. Nice looking HTML page sent back to the user tells him that he has been successfully logged out.

5.7.5 Strengths and weaknesses

This authentication procedure has the following advantages:

- The user password is not given to the faction;
- The faction password is not given to the user;
- The contract ID is a public information that is harmless if used by another party;
- Logs to check vulnerabilities or attacks are easy to implement through contract history; and
- Sessions allow the user to login in a transparent manner from the faction web site.
CHAPTER 5. SOLUTION DESIGN

This authentication procedure raises the following security issues:

- No data encryption is done on the SOAP messages transmitted between the faction web site and the SWCWS;
- Dependability: if one of the server is down, the whole system is down;
- Faction web sites can harm users by doing whatever they like with the character’s data once they have received the appropriate contract.

The authentication procedure provides sufficient security mechanisms to avoid password to get compromised by eavesdroppers sniffing the communications.

NuSOAP is an interesting API, easy to install and to use (it does not require any object oriented knowledge and generates WSDL documents automatically). Sadly it requires PHP CURL extension or OpenSSL to work with SSL encapsulation. This may be an issue for many developers that do not have sufficient access to their web site host to recompile PHP.

5.8 Authorization

The authorization process occurs right after the authentication steps described above. During authentication a sData object is created. This object holds all the authentication information related to the current web service call. It is being passed through the web services component to avoid performance issues related to redundant database connections. The authorization process is using exclusively that object in order to determine if, whether or not, access should be granted. This object will be returned back to allow further use.

```php
/** Secure Data used for Web Services*/
class sData {
    /** key identifying the contract*/
    var $contractID;
    /** password preventing contract guesses*/
    var $contractPWD;
    /** key identifying the client*/
    var $clientID;
    /** character object preventing multiple call to the database*/
    var $char = null;
    /** time the contract was created*/
    var $creationTime;
    /** last time the contract was used*/
    var $usedTime;
    /** client IP address*/
    var $IP;
}```
Two categories of web services access exist. The authorization process has to be able to handle both. The first access category is for web services that are using a client authentication token and do not require any user authentication. The second is for web services that do require a user and client authentication.

A web service requiring client authentication calls the WSpolicy method by passing the string 'client' as an argument. The WSpolicy then runs into the authentication process that builds an sData object, containing all the authentication information related to the connection. Then WSpolicy calls WSauthorize that returns a boolean value true or false depending on whether the connection is authorized or not.

The authorization service is built with a set of rules that are stored on the database and are accessible through the web services management tool, as shown in Figure 5.14.

![Figure 5.14: Rules management tool](image)

Each rule has a type and is uniquely identified by a key, named ruleID. The type of the rule determines its various attributes, such as:
CHAPTER 5. SOLUTION DESIGN

- the character;
- the faction;
- the client; and
- the web service name.

Four types of administrator rules exist:

1. the user-type rules:
   Such rules have the finest granularity. They allow the administrator to authorize or
deny web service access to a particular character using a specified client. A user-
type rule requires that user authentication is used in order to apply.
Example: *Rantan Plan has no access to WSgetFullGNS using client Galactic Empire*

2. the faction-type rules:
   Such rules have a lower granularity as they manage all the members of a faction
at the same time. They are only effective if no user rule was found and if user
authentication is used.
Example: All members of *Galactic Empire* have *access* to *WSgetFullGNS* using client *Galactic Empire*

3. the client-type rules:
   Client rules have a lower priority and granularity than the two previous ones. They
manage all connections made by anyone trying to access particular web services
with a given client.
Example: All user of the *Galactic Empire* client have *no access* to *WSgetFullGNS*

4. the default-type rules:
   Default rules have the lowest priority and granularity of all. They allow the
administrator to set up a default behavior when accessing a web service.
Example: Everyone has *access* to *WSgetFullGNS*

If the rules previously cited in the examples were set up on the server, they would: deny
access to Rantan Plan, but allow access to the members of the Galactic Empire, using the
Galactic Empire client. If Rantan Plan were a member of the Galactic Empire, his access
Chapter 5. Solution Design

would be denied. Everyone using the galactical client would\’t be able to access the service, but they would if accessing it from a different client.

Additionally, from the 4 types of rules set up by the administrators, user-type rules have to be set up by the user. These rules ensure that the user has deliberately allowed a client to access a web service using its account. The way these rules are handled is fairly simple: either the rule exists and access is granted, or it doesn\’t exist and access is denied. Of course this kind of process only occurs when user authentication is being used.

![Figure 5.15: Authorization rules for WS requiring client authentication](image)

Figure 5.15: Authorization rules for WS requiring client authentication

Figure 5.15 represents the different steps the authorization goes through before access is granted or denied to a particular client. The mechanism is fairly simple as it involves only two filters. In case of user authentication, things are a little more complicated, since the user needs to have personally authorized the client to use the web services. Then the authorization process runs through four filters before taking a decision. Figure 5.15 illustrates the mechanism.

![Figure 5.16: Authorization rules for WS requiring user authentication](image)

Figure 5.16: Authorization rules for WS requiring user authentication
Although this system is very powerful since it allows a very fine control of what can be done or not, some negative feedback was received. The user authorization steps were seen as too complicated, as many users were not aware of what web services are involve in most transactions. As an improvement, users proposed the creation of a package concept.

Packages are sets of web services that are working in the same semantic domain. For example, an inventory package was created to group all the web services related to the various assets owned by a faction. For each package, a description is available. Furthermore, for each web service in a package, a user friendly description is given.

The creation of packages didn’t change the core of the authorization service, it was only an improvement made on the user interface of the rules management tool.

5.9 Client implementation

5.9.1 Finding web services

As explained in the technological overview, one important feature of web services is the publication of the remote methods available. This includes what they do, what they require and how to invoke them. While most of this process can be automatically done by WSDL, defining the general purpose of a web service and how it integrates within the application logic remains a developer’s responsibility.

Setting up a UDDI registry would be time consuming and would most likely be too complicated to be frequently used since the number of web services available is still low. Therefore, the decision was made to build a short and easy to use documentation tool, instead of an complete UDDI registry. This tool is very useful for client programmers.

The web service documentation is in two parts. The first part is a graphical interface dynamically generated from the WSDL file and describing what the web services are meant for, how to call them and what they return. The second part is a web service that lists all the web services known by the server and that are accessible with a specific security token.
5.9.2 Implementation issues

During the development cycle, a huge attention has been paid to feedback and communication with the web services early adopters. The following authentication issues have been raised during those discussions.

- Client applications not running behind a web portal and not using any browser technology require a way of logging in. Two options are available to them.
  1. Instead of requiring a user name and password, they require a contract number. The user has to connect to the provided web page to retrieve the adequate contract. This solution is very secure, as it respects the authentication paradigm explained previously and does not require any modification. It is therefore the preferred method.
  2. The client authenticates and gets a client security token. Using that token, it can access a web service that receives a user name and password and that allows it to upgrade its client security token to a user security token. This user authentication mechanism is very convenient for users. However, it presents a major drawback: the user password needs to be typed in the client application and therefore could be easily compromised.

- Some clients would like to receive special privileges to access some functions. Indeed, the only way to access a protected game feature is to authenticate as a user. But factions would like to be able to authenticate as such. This kind of special privileges would be very useful, for example, when a client wants to send an notification to a user through the SWC messaging system. Very easy to implement on the web services infrastructure, this problem is still open, as it involves game mechanisms and traceability issues concerning the author of a message.

5.9.3 Examples

This section holds the various client prototypes I wrote to prove the technical feasibility and to motivate developers to create their own.

**PHP client**

The PHP client was made as the counter part of the server side developed under NuSOAP.
The Java client has been built to demonstrate that the web services created are fully independent from the platform and language on the client side. The creation of a proxy is an automated task under Eclipse 3.1. As discussed previously, the WSDL file is parsed to allow the creation of the SOAP messages. Although it is working, the WST is still under development and the configuration is very tricky.

```java
get newCGT = null;
try {
    newCGT = proxy.USgetCGTObject();
    proxy.USgetCharProfile(newCGT, "Veynon");
} catch (Exception ce) {
    try {
        System.out.println(ce);
    } catch (Exception e) {
    }
}
finally {
    if (newCGT != null) {
        theCGT = newCGT;
    }
}
```

---

3Web Service Toolkit is a set of packages allowing Eclipse to connect Web Services.
CHAPTER 5. SOLUTION DESIGN

The Java console application connects to SWC and returns the server time. Very short and easy to write, it demonstrates how the WST tools make it convenient to write web services, once the IDE configuration has been set up.

C# client

Visual Studio .NET has full web services support allowing fast and easy SOAP clients creation.

![C# client](image)

(a) Login page  (b) Inventory Listing

Figure 5.19: C# client

```csharp
public string browseProfile(string handle)
{
    combine.WSitCharProfile(sToken, handle);
}

public Form1 getMainForm()
{
    return this.myForm;
}
```

Figure 5.20: Integration in Visual Studio 2003

The C# client offers a graphical user interface that lists the various inventories that belong to a user. It fully complies with the Model-View-Controller paradigm. It could be improved by adding more features and data caching.
Chapter 6

Results

6.1 Clients using web services

As explained in the last chapter, the web services offered by the SWC can be grouped into three categories, according to their access level:

- public web services:
  These web services do not require any kind of authentication. Their purpose is advertisement. They are really easy to implement but don’t offer a lot of functionalities. They can be used to obtain server time, time conversion and the list of all visible systems in the galaxy or estimate time of arrival for a given travel;

- client-restricted web services:
  These web services require only client authentication. They have more functionalities than public web services: clients can download the whole galactic map, browse the character profiles or connect to the galactic news system;

- user-restricted web services:
  These web services require both client and user authentication. Players can download their character information, list the faction members, connect to their personal inventory or retrieve their faction assets;

To get access to both client- and user-restricted web services, developers need to register their client vis-à-vis the SWC administration. So far, ten clients went through the registration process successfully.
CHAPTER 6. RESULTS

The Galactic Empire is one of the largest faction in the game. As any other faction, it has its own web site. It uses PHP to provide dynamic content to its members. It connects to the SWC server through web services to perform automated tasks.

Over the years, the Empire has developed its own news system. Thanks to web services, the web portal is now capable of merging its own news with the SWC news. Once sorted in chronological order, the news are displayed to the members. This system provides to all the members of the Galactic Empire a single and user-friendly web page containing all the galactic news. The web page is shown in Figure 6.1.

The CenterPoint Space Station, another large faction, is using web services to keep its inventories up to date. Once a player has performed authentication through the web services mechanism, he has access to all his character’s belongings and he can also obtain a list of his faction’s assets, provided he has the required privileges. Figure 6.2 shows the web page generated on the CenterPoint Space Station web portal.

Another very impressive achievement made by the CenterPoint Space Station is the creation of a dynamic map to locate its assets in the galaxy. Assets are marked with dots on the map. When moving the mouse to one of these dots, the user can obtain various information about the assets placed there.
6.2 Statistics

Every time a client is successfully authenticated and authorized to use a web service, a record containing its clientID, the userID, the time and the web services requested is stored on the database. All these data are then used for statistics purposes. Figure 6.4 shows the web services use from 16 June to 8 August 2005.
Figure 6.4: Web services usage

As the Figure 6.4 shows, a number of clients have already started to use the web services, although the latter are still in their infancy. Not all clients are interested in the same features.

The Galactic Corporation is the eBay version within the SWC universe. One of its main activity consists of an on-line auction web site for objects in the galaxy. It is concerned with many security issues, in particular the verification of its user accounts. Not surprisingly, the Galactic Corporation has mainly been using account verification web services.

The CenterPoint Space Station is mainly interested in web services providing inventories of ships, space stations, raw materials, etc. The CenterPoint Space Station is now able to call upon other web services related to the galactic map, such as WSgetSystems or WSgetPlanets. Indeed, they have just finished programming their client accordingly and players will soon take advantage of this new application. Therefore, an increase in the use of these web services is to be expected in the near future.
CHAPTER 6. RESULTS

One should also notice the huge amount of calls made by the Galactic Empire to WS-getFullGNS in order to access the news system. This identifies a potential security hole. Therefore, security measures have to be taken to avoid massive resources consumption by a minority of clients.

An automated limitation tool will need to be put in place so as to avoid abuses or unfortunate incidents. In this very case, the Galactic Empire was unaware of the large number of calls made to web services.

6.3 The community acceptance

One major objective of this project is to observe how the SWC community responds to the creation of the web services infrastructure. Would it be worth to continue developing such infrastructure outside of the academical context?

Measuring community acceptance of new features, such as the web services proposed in this paper, in a massively multi-player on-line role-player game is a difficult task. The measurement includes many elements, which need to be analyzed by various specialists such as psychologists, sociologists, game experts, etc.

However, some indicators predict a good future for a wider adoption of web services in the MMORPGs context.

Some of these positive indicators are:

- The large number of clients developed or being developed over a very short period of time;
- The numerous complaints about the imperfections in the pilot web services architecture proposed;
- The huge number of requests coming from the developers for new web services features;
- The high number of utilizations revealed via the statistics tool;
- The initiative taken by some developers to create a wiki web site dedicated to document the web services component and share their code; and
• The great speed at which bugs have been identified.

Some doubts have been expressed about the future of the use of web services in SWC:

_Unfortunately Web Services will only be able to be used by those with necessary development experience and the ability to implement them into their faction website. With various levels of coding experience amongst the owners of faction sites it is impossible to predict whether or not web services will be used to their full. Whilst the ability is their to use them I am afraid that much of it will go to waste._

–Zorran Black Myorzo Corporation founder

However, the following quotes express positive support:

_As a developer and user, I’ve first looked at WS with a little skepticism, but after the first stages, I’ve got quite fond of them. With use of the Web Services, I’ve been able to greatly enhance the available features on my faction’s website, as well as create completely new ones. With the current development, who knows what i’ll be able to do next._

–Togan Jano
CenterPoint Space Station Developer

_Web services are a promising feature for the Galactic Market and Galactic Corporation’s other ventures. Expansion into asset authentication, where items put up for sale are checked to see whether that person has the permissions necessary to sell them, or whether they exist in the first place, may be implemented. Other areas such as unified logins with other SWC community sites may also become a possibility. Many things are possible with Web services, but you can rest assured that any steps taken will be taken with consideration for you, the user._

–Niels
Galactic Market Developer
I like web services a lot. I spent a fair amount of time developing them in PHP while writing the basics of my client, and it was really easy to work with all of the functions (but I later found that not to be the case, in Java). Java, however, was painful to use SOAP through because of its ridiculous class structure which required the creation of a million objects that serve no purpose other than to house the object that you create from them.

Conceptually, Web Services are an excellent idea and would work wonderfully with a good core set of functions/classes, because it would amount to creating a new interface. However, there is a serious trust issue by allowing people to create their own web services client. We (Jolly and I) talked about this for some period of time, but we can’t seem to come up with any way to allow users to manipulate assets and trust them to only execute transactions allowed by the user. Given the various ways that we came up with for authenticating those transactions, it would probably be better for users to simply use the interface provided by SWC.

Despite that simple drawback (which is more inherent of the system we are manipulating through WS, rather than the actual WS), they are a powerful tool for user account authentication and are definitely a nice way to help other sites from SWC link into it, even if it is only because they can allow you to read your messages, see the GNS, or see SimNews. It still feels like SWC away from www.swcombine.com. That simple link created by WS makes the entire community seem more like one object, rather than several.

Regarding the difficulties coding WS, Jolly provided all the PHP people with a file that would create the connection, cache the wdsl, and create an object that one could use to call all of the WS functions. I don’t see how it was ‘more difficult’ because all that had to be done was add a line including this file, and then one had to be familiar enough with OOP to use the function calls. But, PHP support among hosts is a bit of an issue, so I could see that as a drawback. The only real drawback.

–Selatos

PHP GTK client Developer
I would like to, on behalf of the Galactic Market, its users and staff, thank Jolly for the work he has done on the new Star Wars Combine Web Services project. His additions to our arsenal against scammers and thieves has been invaluable. For that, we are very thankfull.

Thanks again, Jolly, keep up the excellent work!

–Niels

Assistant Sim Master

Star Wars Combine

“The first time I heard about Web Services, I simply thought: "Ok, it’s a cool concept." and I stored this in a back corner of my memory. A few months later, I was discussing with Star Wars Combine players about how the game stopped being a game and started being a second job for many veterans. Investigations demonstrated that most of the time, veteran players reached important in-game positions where they had to manage factions and players. This management (keeping tracks of assets, of members and their role...) proved to be extremely exhaustive and could require several hours per week. To automate some of the management process, these players created dynamic websites relying on databases to copy game information. While this solution helped them at first, it quickly forced them to update to data systems: the game, where they play and perform actions and then their own website. Quickly, a question became unavoidable: how to prevent this duplicate work so that these players can spend their time in having fun rather than in managing data.

Then I remembered Web Services were doing exactly that: transferring data from a machine to another. As the Star Wars Combine lacked the necessary resources for this exploratory project, I proposed the idea as topic for a Master thesis to the University of Brussels, and it got accepted.

A year later, we can see the first effective results. Web Services automate the processes, prevent data discrepancies, and reduce data management. Furthermore, some motivated players could use the retrieved data
to create enhanced applications for their faction: some created automatic checks, some created geographical inventories, and so on. In their way, Web Services are a definitive advantage for the remote management of a persistent universe faction. However, their principal drawbacks remain the technical skills that the end-users have to master to implement them and the additional development and maintenance resources they require.

As I write this, the Star Wars Combine is still a pioneer concerning the usage of Web Services for the management of MMOG’s factions. Their quick development generated various questions concerning this use like "What will be allowed through our Web Services and what will not ?" but it seems they eventually earned their place into the Star Wars Combine concept.”

–Veynom

Sim Master

Star Wars Combine
Chapter 7

Conclusions

The objectives of this thesis were:

1. to imagine a web services architecture for SWC;
2. to integrate the architecture into the already deployed production server;
3. to demonstrate the usability and added value of the technology; and
4. to measure the technology acceptance among the whole SWC community.

After a brief overview of the game concepts carried by SWC and the web services technologies involved, we took a look at the various advantages such services could bring to the game. In collaboration with the game developers, specifications for the web services component were written and an integration roadmap was drawn up: the new components would be standing in parallel to the existing Graphical User Interface, below the library layer that holds the game engine and accesses the database.

Development and integration of the web services component were done iteratively. The first step was to create a basic architecture allowing public web services to be integrated in the current game. New components such as authentication, authorization and documentation tools were progressively added. At the same time, new web services were introduced to increase the functionalities offered and the interest of the faction programmers.

After nine months of iterative development, web services reached the stage of direct trust relationship defined in chapter 2. The quite impressive number of clients developed in a very short time has demonstrated the usability of the technology. The current release
of the SWC web services component allows clients to retrieve information about inventories, characters, galaxy map, galactic news and time as structured data easily integrable in any other applications.

While building the web services server application and proving its usability and usefulness was quite an achievement, the community acceptance objective was not forgotten. I think this objective was also reached, as demonstrated by the comments received from users. All the primary objectives listed above were successfully met, mostly thanks to the excellent communication between the community and the developers, including myself. To allow the project to continue independently from its developers, manuals are provided. They are accompanied with an automated documentation tool and a carefully commented code.

As for the future of the web services component, I strongly believe that it can easily be expanded to include new features. As the infrastructure is now in place, this would not take up much time. On the other hand, it would be very useful to set up a developer working group to insure that the new features are in line with the game rules.

Concerning the development of the next version that would allow to reach the indirect trust relationship stage, I, however, recommend to pay great attention to the following aspects:

- the important amount of time which will be needed to set up the necessary public key infrastructure;

- the client technical limitations, as most clients will not be allowed to install the cryptographic libraries required; and

- the high level skills that will be requested from the developers, in particular on the client side.

For these reasons, I don’t recommend that the current web services component be upgraded in the short time. SWC is already a pioneer with regard to web services: no other game is yet using such a technology. I think it would be wise to wait for more feedback from the community before investing more time in web services.

I have enjoyed working on this project. It taught me a lot. I’d like to thank all the people that made it both a technical and personal success.
Appendix A

Web services user manual

A.1 Setting permissions

Before being able to use the web services functionalities, a user has to allow the client to access the web services. Users do so by loading the web services management page: “Members->Manage->OOC Profile->Web Services”

The user has to select sets of web services and client applications. For each web service belonging to a set, the selected client will be granted an access.

![Figure A.1: Setting user permissions](image)

Users can also delete web services by selecting those in the list, and clicking on the delete button. No more access will be granted to the selected web services.
A.2 Setting auto grants

When a user authenticates himself, a warning message appears to make sure he is aware that a connection is being established. He is asked to grant connection access before any web service can be executed on the client side. To make the authentication process more transparent, he can select the auto-grant option. With the auto-grant option selected, the warning message won’t appear anymore and further connections can become completely invisible to the user.

![Auto-grants list](image.png)

Figure A.2: Removing an auto-grant

The auto-grant management tool allows a user to remove the auto-grant permission he has previously set for a particular client.
Appendix B

Web services client manual

B.1 Client development

Creating a proxy

The proxy is an object through which all the remote methods are accessible. All the methods are created dynamically by loading and parsing the WSDL file made available on the server.

```php
<?
// modify this line to point to your nusoap lib
include_once('nusoap/nusoap.php');

// create the client

// exit if error while creating the client
$err = $client->getError();
if ($err) {
    echo "<h2>Constructor error</h2><pre>" . $err . "</pre>";
    exit();
}

// SWCWS is a proxy for the client
$SWCWS = $client->getProxy();
?>
```

Creating a proxy with caching

To reduce the load, it is also possible to cache the WSDL file. This avoids the server to generate a new description and to transmit it.

```php
<?php
////////////////////////////////////////////////////////////////////////////////
// CONFIGURATION
```
Your first web services call

Let’s try to make a very simple Web service call that returns the galactic time.

// include the caching proxy
include_once("wsapi.inc");
// call the remote function
$cgt = $SWCWS->WSgetCgtObject();

// save the soap messages
$soapmessages =
  '<h4>Request CGT</h4>' .
  '<pre>' . htmlspecialchars($SWCWS->request, ENT_QUOTES) . '</pre>';

$soapmessages =
  '<h4>Response CGT</h4>' .
  '<pre>' . htmlspecialchars($SWCWS->response, ENT_QUOTES) . '</pre>';

// detect any soap error
if ($SWCWS->fault) {
  $output = "ERROR ! "$SWCWS->faultstring."\n";
}

// handle the received data
else {
  $output = 
    "year:" . $cgt['years'] . 
    " day:" . $cgt['days'] . 
    " hour:" . $cgt['hours'] . 
    " min:" . $cgt['mins'] . 
    " sec:" . $cgt['secs'];
}

// echo the output
echo $output;

Getting a security token

Most web services, however require a security token (sToken) as parameter in order to work. This sToken provides user or client authentication, depending upon the contract provided by the user.

define('WSclientID',12345);
define('WSclientPWD','acbd1234');

include_once("wsapi.inc");

session_start();
// sToken saved ?
if (!isset($_SESSION['sToken'])) {
  // contract received -> user authentication
  if (isset($_REQUEST['WScontract'])) {
    $_SESSION['sToken'] = $SWCWS->WSgetsToken($_REQUEST['WScontract'],
      $WSclientID, $WSclientPWD);
  }
  // no contract received -> client authentication
  else {
    $_SESSION['sToken'] = $SWCWS->WSgetsToken(}
Using a security token

By using a security token as the first parameter to the web service, a client can access restricted web services. It should save the security token as it can be reused to call other non-public web services. The following code illustrates how to call a non-public web service.

```php
$output = "<h3>Read last message from inbox (WSgetMessages)</h3>";
$status = $SWCWS->WSgetMessages( 
    $_SESSION['sToken'],
    0,
    1,
    "inbox");

$soapmessages .= '<h4>Request get messages</h4>' .
    '<pre>' . htmlspecialchars($SWCWS->request, ENT_QUOTES) . '</pre>';
$soapmessages .= '<h4>Response get messages</h4>' .
    '<pre>' . htmlspecialchars($SWCWS->response, ENT_QUOTES) . '</pre>';

if($SWCWS->fault) {
    $output .= $SWCWS->faultstring."\n";
} else {
    $output .= '<pre>'.var_export($status,true).'</pre>';
}

echo $output;
echo $soapmessages;
```

Destroying a security token

Once the client has finished with a security token, it should be destroyed. The following example illustrate how to destroy a security token.

```php
$status = $SWCWS->WSlogout($sToken);
if($SWCWS->fault) {
    $output .= $SWCWS->faultstring."\n";
} else {
    if ($status == true) {
        unset($sToken);
        echo "security token successfully destroyed !";
    }
```
else {
    echo "ERROR: security token not destroyed!";
}

}
B.2 Improving integration

To further improve the integration between your web site and the login page, the client can specify a style sheet file for the login page. The client need to register by a web services administrator the css URL it wants to use.

The following figure is an example on how the login page can be modified using CSS.

![SWCWS Login](image)

Figure B.1: CSS example

You can also force the redirection from the login page to another page than the default one, you have registered during client registration. To do so, simply pass a WSredirect variable with the file name through HTTP GET method.

For example, the following HTML code redirects the user to the page named: “test.php”:

```
```

You can also specify the complete URL of the redirection instead of the file name. Be aware, however, that the redirection will only work if there is a match with the domain name you have specified during the client registration.
C.1 Web services development

Linking with the game libraries

```php
// load the server configuration file
include_once("/var/secwww/path.php");

// load the ws authentication / authorization modules
include_once(WEBSERVICES_PATH.'/lib/sessions.inc');

// load additional libraries
include_once(LIB_PATH."/characters.inc");
include_once(LIB_PATH."/com.inc");

/**
 * Get messages for specified receiver. Only returns unhidden messages.
 * Returns Collection of Message Objects with following accessors:
 * - transactionID
 * - time
 * - communication
 * - sender
 * - receiver
 * - image
 * @author Jolly
 * @param sToken $sToken user sToken
 * @param int $intStart value to start retrieving messages from.
 * @param int $intLimit maximum messages we want to return.
 * @param string $strBox name of the box {"inbox","outbox"} default is inbox.
 * @return collection
 */
function WSgetMessages($sToken = null,
        $intStart = 0,
        $intLimit = 10,
        $strBox = 'inbox') {
    $sData = WSpolicy($sToken,'user');
    // receiver type 1 = character
    $intReceiverType = 1;
    switch ($strBox) {
        case "outbox":
        case "sent":
```
Function registration

include_once("/var/secwww/path.php");
include_once(WEBSERVICES_PATH.'/nusoap/nusoap.php');
include_once(WEBSERVICES_PATH.'/lib/com.inc');

// create default endpoint if no endpoint has been specified yet
if (!isset($server)) {
    $server = new soap_server;
    $server->configureWSDL('swcws',WEBSERVICES_ENDPOINT);
}

// load objects
include_once(WEBSERVICES_PATH.'/registration/objects/sToken.inc');
include_once(WEBSERVICES_PATH.'/registration/objects/messages.inc');

$server->register('WSgetMessages', // method name
    array(  // input parameters
        'sToken' => 'tns:sToken',
        'intStart' => 'xsd:int',
        'intLimit' => 'xsd:int',
        'strBox' => 'xsd:string',
    ),
    array(  // output parameters
        'return' => 'tns:MessageCollection', // namespace
        'urn:swcws', // soapaction
        'urn:swcws#WSgetMessages', // style
        'rpc', // use
        'encoded', // use
    ),
    'Retrieves message for specified receiver. Only returns unhidden messages.
    @author Jolly
    @param sToken $sToken user security Token
    @param int $intStart value to start retrieving messages from.
    @param int $intLimit maximum messages we want to return.
    @return collection'
);

Object registration

$server->wsdl->addComplexType(
    'message',
    'complexType',
    'struct',

88
'all',

array(
    'transactionID' => array('name' => 'transactionID', 'type' => 'xsd:int'),
    'time' => array('name' => 'time', 'type' => 'xsd:string'),
    'communication' => array('name' => 'communication', 'type' => 'xsd:string'),
    'sender' => array('name' => 'sender', 'type' => 'xsd:string'),
    'receiver' => array('name' => 'receiver', 'type' => 'xsd:string'),
    'image' => array('name' => 'image', 'type' => 'xsd:string')
)
);

$server->wsdl->addComplexType(
    'MessageCollection',
    'complexType',
    'array',
    ',
    'SOAP-ENC:Array',
    array(),
    array(
        array('ref' => 'SOAP-ENC:arrayType',
            'wsdl:arrayType' => 'tns:message[]')
    ),
    'tns:message'
    );
C.2 Web services administration

C.2.1 Client registration

The current registration process requires the client to send an email to the administrators with the following informations:

- the client name;
- the main developer’s real name;
- the main developer’s user name;
- the main developer’s email address;
- the faction name the client is developed for. If the client is used for personal use this field can remain empty;
- the URL of the CSS style sheet to be used for the login page;
- the URL of the web portal.

The database is filled accordingly. All the information is kept in the table named “WSclients”.

Once registered, an email is returned to the developers specifying the client ID and the password to be used in order to get a valid security token.

C.2.2 Permission settings

Once a client is registered, the administrators can control what web services it will have access to, by using the rules administration tools located in: “Members->Manage->Administration->Web Services”.

Using the graphical interface, the administrator can add or delete the four types of rules as described in the authorization section of chapter 5.

To delete a package, the administrator simply selects from the corresponding “check box” and clicks on the delete button.
C.2.3 Packages administration

Packages are set of web services that are grouped together with the same semantic. This is meant to make package selection easier for the users.

The package management tool works in a very similar manner to the permission management tool. To add a package, the administrator has to fill in a form specifying the package name, the package description and the web services it will contain. To delete a package, the administrator selects the corresponding “check box” and clicks on the delete button.
Bibliography


[12] Mike Downen and Shawn Farkas. Exchange data more securely with xml signatures and encryption.  


[21] Dr. Frank Leymann. Web services flow language (wsfl 1.0).  


[23] OASIS. Oasis security services (saml) tc.  


[26] Liberty projects. Liberty specs tutorial.  


http://www.w3.org/Style/


[34] Inc XMethods. Xmethods.  
http://www.xmethods.com/