STANDARDS FOR COMMUNICATION AND E-LEARNING IN VIRTUAL WORLDS

The Multilingual-Assisted Chat Interface

Samuel Cruz-Lara, Tarik Osswald, Jordan Guinaud, Nadia Bellalem, Lotfi Bellalem
LORIA / INRIA Nancy - Grand Est, 615 Rue du Jardin Botanique, 54600 Villers-lès-Nancy, France
{*samuel.cruz-lara, tarik.osswald, jordan.guinaud, nadia.bellalem, lotfi.bellalem*}*@loria.fr*

Keywords: Multilinguality, MLIF, Chat Interface, Web Services, Communication, E-Learning, Automatic Translation, Virtual Worlds

Abstract: Many of today's applications embed textual chat interfaces or work with multilingual textual information. The Multilingual Information Framework (MLIF) [ISO DIS 24616] is being designed in order to fulfill the multilingual needs of today's applications. Within our research activity for the MLIF standard, we developed the Multilingual-Assisted Chat Interface, which intends to help people communicate in virtual worlds with others who do not speak the same language and to offer new possibilities for learning foreign languages.
By developing this application, we also wanted to show the advantages of using web services for externalizing computation: we used the same web service for two virtual worlds: Second Life and Solipsis.
In this paper, we first propose a short analysis of social interactions and language learning in virtual worlds. Then, we describe in a technical way the features, architecture and development indications for the Multilingual-Assisted Chat Interface.

1 INTRODUCTION

Today, talking to people via a textual chat interface has become very usual. Many web applications have an embedded chat interface, with a varying array of features, so that the users can communicate from within these applications. A chat interface is also easy to implement: unlike voice or video, it needs neither additional devices nor additional signal analysis algorithms. Therefore it is not surprising if brand new technologies such as virtual worlds also embed a textual chat interface.

But all applications have their own peculiarities, and their chats also serve various requirements. A distinctive feature of virtual worlds is that people are more likely to converse with other people who cannot speak their native language. In such cases, the need for some sort of assistance in facilitating interlanguage communication becomes obvious. A chat interface with multilingual features can meet these requirements.(Cruz-Lara et al., 2009). Moreover, such an interface can be turned into an advantage for people who want to improve their foreign language skills in virtual life situations.

In this paper, we first want to present some considerations about social interactions and language learning in virtual worlds. Then, we will describe chat interfaces in general and more specifically the Multilingual-Assisted Chat Interface, which we have developed within the ITEA2 Metaverse1 (www.metaverse1.org) Project [ITEA2 07016] in order to give a first answer to the question "how can we ease communication in virtual worlds and turn the multilinguality issue into an advantage?"

2 SOCIAL INTERACTIONS

The social interaction in virtual worlds constitutes a particularly rich field of research. Indeed, the objective to be reached is to offer simple and effective means of communication, which approach the natural communication. In this context, efforts should be made as much in improving the technical aspects as in taking into account the socio-cognitive aspects. Thus, it will improve the realism of the virtual envi-

ronments and increase the quality of the exchanges between the avatars. Therefore, gestures, speech, direction of gaze, postures or facial expressions have an entire role in the construction of the social links between individuals.

The question concerning the influence of virtual environments on the social behaviour of users, constitutes a particularly interesting topic for the researchers in sociology. In fact, they highlighted a phenomenon of disinhibition and facilitation which leads to a greater sociability (Suler, 2004)(Coleman, 2007). As social human beings, we adjust our behaviour with the social norms in face to face communication. We know, thanks to our education and our culture, what is socially acceptable or not. Communicating by interposed computers strongly decreases this adjustment because we cannot observe in real-time the effects of our words and of our writings.

In order to propose credible and powerful communication between avatars, the characteristics of the human communication must be taken into account: language with explicit or implicit references to the objects of the environment, gestures, postures, facial expressions. These elements of communication are all the more important since the avatar is immersed in a three-dimensional world populated by objects, personages and places more or less characteristic which sometimes echoes the real world.

The models suggested by El Jed (El Jed et al., 2006) try to take into account intentional communication as well as non-intentional communication in the interpretation of the acts of communication between avatars. In this context, the favoured mode of communication is natural language combined with deictic gestures. The difficulty, in this case, consists in using markers like vocal intensity, voice intonation, or indexical or deictic references ("I", "here", "over there") associated with the designation gesture to determine the relevant interpretation of the exchanges. The direction of the gaze can also be exploited in order to focus the visual attention of the interlocutors towards a specific place in the shared environment. The facial expressions are essential during the exchanges and constitute the first channel to communicate emotions. They can express mood, approval or disapproval, but also the whole panel of the human emotions (fear, joy, etc.). All these manifestations of the human communication would be very useful in the domain of education, specially for the development of e-learning techniques used for the realisation of virtual campuses (De Lucia et al., 2009).

3 LANGUAGE LEARNING

Language learning in virtual worlds is a new field of research which is still open to innovations. How can we create technological advances in order to create an optimal psycholinguistic environment for language learning? What makes new proposals innovative and helpful? In order to answer these questions we are currently developing some empirical support.

We believe that one potential source of guidance may be offered by some methodological principles of Task-Based Language Teaching (TBLT) applied to distance foreign language teaching (Doughty and Long, 2003). It should be noted that the TBLT aspect of our work is currently under development. The general idea is allowing teachers to create TBLT units via the Moodle learning environment system (http://www.moodle.org), and then to use SLoodle (http://www.sloodle.org). SLoodle is an open source project which integrates Moodle components in Second Life.

Our approach (i.e., the Multilingual-Assisted Chat Interface) must be considered as an innovative form of Computer-Assisted Language Learning (CALL).

4 THE MULTILINGUAL ASSISTED CHAT INTERFACE

The Multilingual-Assisted Chat Interface is a tool offering new functionalities to the chat users in Virtual worlds. It is directly embedded in some virtual worlds viewers (to date: Second Life and Solipsis).

In order to implement those functionalities (described in Section 4.2), we modified the source code of the viewers (see Section 4.5).

The Multilingual-Assisted Chat Interface mainly relies on the Multilingual Information Framework (Cruz-Lara et al., nd), which is a standard being currently developed by the International Organization for Standardization (ISO).

4.1 Generalities about chat interfaces

The term "chat" refers to a real-time written dialogue, using a computer. The chat presents many similarities with the oral dialogue. In fact, it surely is the written means of communication which is the closest to it. Indeed, it is closer than other tools, such as forums and email.

Most of the time (for example in Second Life), the chat is volatile. In fact, the contents of a chat session, like an oral conversation, is not intended to be available to the public. Furthermore, when a user logs in,

he totally ignores what has been said before his arrival. Similarly he cannot know what will be said after his disconnection. Moreover, the chat users introduce into their writing some elements which are specific to oral communication.

The chat is usually based on a client-server architecture, meaning that users do not communicate directly with each other, but through a single server. All the chat users connected to one server do not necessarily communicate together. In general, a server gives access to several channels of discussion (also known as rooms), which are completely partitioned. The chat users only see what is happening in the channel they are connected to, and they can only send messages to this very one. On most of the public servers, the creation of channels is completely free, and any user can open some.

In general, the client chat interface is split into three areas:

- the ongoing conversation;
- the message editing and sending zone;
- the listing of the connected users.

A chat session can be considered as a set of messages, ordered sequentially and produced by various authors, humans or robots.

4.2 Functionalities

In this section, we are going to describe the three main functionalities which we implemented in the Multilingual-Assisted Chat Interface.

4.2.1 Grammatical analysis and word colouring

This is the first feature that we implemented. It consists in coloring some specific words in a sentence written in the chat interface, in order to show the grammatical structure of the sentence more clearly. The settings can be modified in a specific menu: the user can choose which grammatical categories they want to highlight, and the corresponding color. They can also choose which sentences they want to analyze (other people's sentences, objects' sentences, all the sentences, etc.).

The grammatical part-of-speech analysis is performed on a remote web server, using the method described in Section 4.4. The data structure used for tagging the grammatical category of each word is MLIF.

4.2.2 Providing word translations, synonyms and definitions

An important feature is the ability to simply click on a word in the chat interface in order to get definitions, synonyms and/or translations of this word. In the current implementation, the definitions and synonyms are retrieved from WordNet, and the translations from Google Translate. It is important to note that other external corpora could also be used.

This feature could be needed in two main situations:

- When people are reading text in a language they do not know very well, they may need some help (*e.g.* definitions, synonyms, translations) in order to understand even very common topics.
- When the discussion is about a rather precise subject, where several technical terms are often used, people may especially need additional definitions even if they are native speakers.

The obvious advantage of this feature is that it is able to aggregate information from several web services in just one click, which is much more convenient than looking for the information in a web browser by oneself. Figure 1 shows the menu, the chat interface (with the verbs in a different colour) and the action performed when clicking on a word.

Every MLIF data structure generated after clicking on a word can be stored in a database. Doing so provides the user with a feedback on the words that he did not understand that well. Writing down the unknown vocabulary is something very usual when learning a new language and this is exactly what we have implemented here.

4.2.3 Automatic translation

When people do not know a language very well, they will be interested in having an automatic translation of every sentence. In this case, it is no longer an e-learning functionality but a way to make communication easier between people in the virtual worlds.

For example, user A uses the Multilingual-Assisted Chat Interface and wants to have all the messages displayed in French. The Multilingual-Assisted Chat Interface will analyse the incoming messages and translate them if they are not written in French (using Google Translate). Then, the messages sent by user A will be translated with respect to the language of the latest received message in the conversation. Figure 2 shows a typical situation involving two avatars who neither understand nor speak the same language.

Another important point is that the source language is automatically detected by Google Translate. Thus, the user only needs to enable the automatic translation functionality to be able to chat with any other avatar in the virtual world. Also, as the source language is stored in the MLIF data, it is possible to

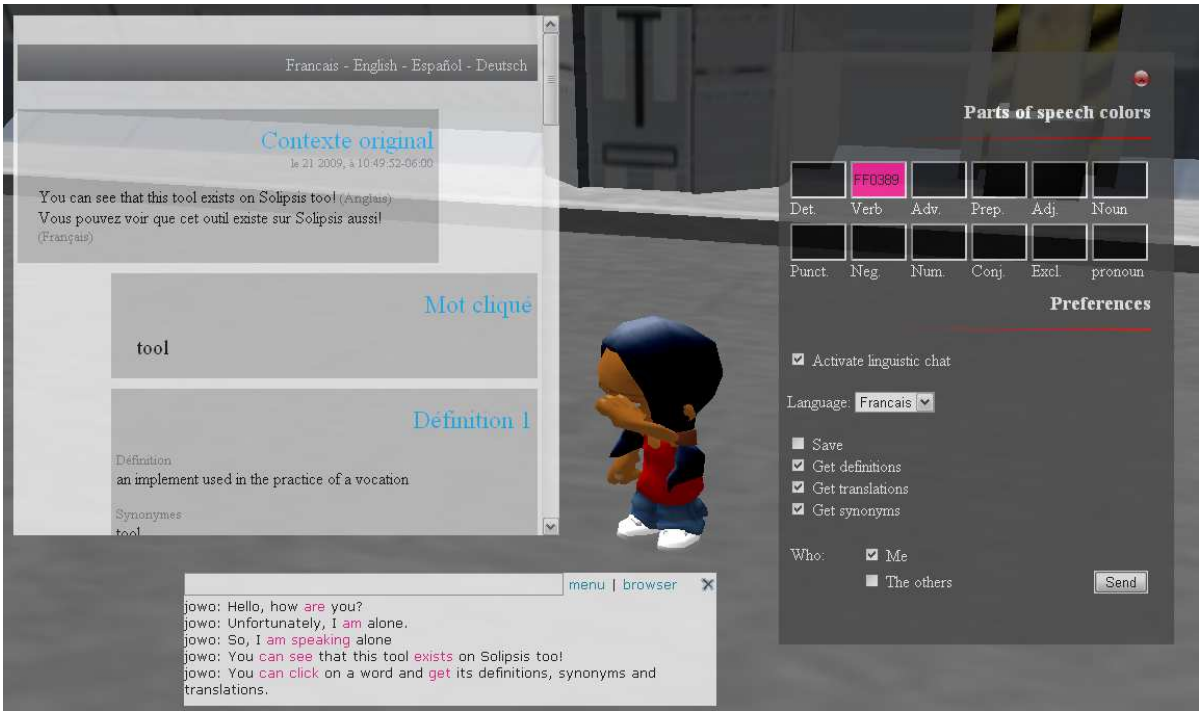

Figure 1: Colouring words in the chat interface and displaying information about one word (Solipsis)

Figure 2: Automatic translation between a Japanese and a French avatar

link every discussion to a pair of languages and as a consequence, it is possible to carry out several multilingual conversations at the same time.

In addition to that, both messages are stored as MLIF data. Thus, the user can click on one translated sentence and see the original sentence if they want some insight into the original language or if the translation does not seem very accurate.

4.3 General Architecture

Figure 3 shows the general high-level architecture of the main components of the Multilingual Assisted Chat Interface.

Three colors are used in this scheme. Each one represents a certain category of components:

orange (on the left): components belonging to the virtual world (especially the viewer);

blue (in the middle): the web service components that we developed, mainly business components;

green (on the right): external web services and corpora, and data storage.

The circled numbers represent the chronological order of the interactions between the components when a message is sent or when a word is clicked. The corresponding explanations are written below:

1. Every message sent by a user is first sent to the virtual world server. When the client (i.e. the viewer) of the person we are writing to receives a message, this is forwarded to a *Message Manager*, (i.e. the component set dealing with the chat messages). We needed to modify these components both in Snowglobe (a viewer for Second Life, see Section 4.5.1) and in Solipsis.
2. Before displaying the message on the *Chat Interface*, the *Message Manager* sends an HTTP request to the web service (the *Grammatical Analyser*) in order to obtain the MLIF data representing the sentence and its several grammatical components.
3. The *Grammatical Analyser* connects to external *Grammatical Corpora* in order to get the grammatical part-of-speech tag for each word of the message.
4. When the *Message Manager* receives the MLIF data structure representing the original message with a grammatical labeling (as an HTTP response), the MLIF data is parsed and turned into a format enabling the coloring and hyper linking of each word. The coloring code depends on the settings of the user interface (see Section 4.2.1).
5. An action is performed: while the addressee reads the message, they click on a word that they do not understand (in the *Chat Interface*) in order to retrieve synonyms, definitions and translations for this word.
6. After clicking on the word, the user is redirected to a *Web Interface*, which will display all the desired information.
7. Loading the web page involves calling the *Word Request Manager* so that it retrieves information from external web services.
8. First, the *Word Request Manager* retrieves some definitions and synonyms from *WordNet*.
9. Second, the *Word Request Manager* connects to *Google Translate* in order to retrieve translations.
10. Once the *Word Request Manager* has received all the desired information, all of it is written into an MLIF data structure. This MLIF data is stored in a database, from which it can be retrieved later if required (see Section 4.2.2).
11. The MLIF data is then sent to an *MLIF-to-HTML Parser*, which is going to transform the MLIF data into user-friendly HTML code.
12. The HTML code thus obtained is finally displayed by the *Web Interface* user interface, which makes it easy for the user to read.

It is important to note that most data exchanges are made using MLIF, since this data format exactly fits our requirements. This is very important to facilitate the development within the web service (as we only use one data format) and for future applications (as it enhances interoperability).

4.4 Web Services For Virtual Worlds

The Multilingual-Assisted Chat Interface is based on a web service. While the virtual world client viewer is in charge of displaying information, the web service deals with data processing.

The text is sent by the viewer (Second Life, Solipsis, etc.) to the web service. The latter gathers the information requested by the users and puts it together into a MLIF data structure. Finally, the generated MLIF data is returned to the viewer, which turns it into a displayable format.

The web service has a grammatical tagger tool (Brill, 1992). It is composed of two Python-generated dictionaries: a default tag dictionary and a rule dictionary. The first one contributes to the matching of each word with its most likely category, the second one to the correction of errors by checking the context.

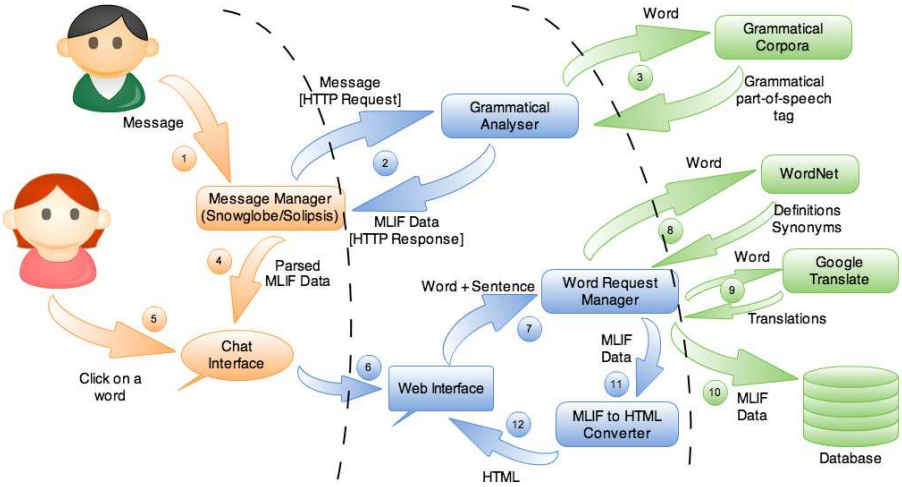

Figure 3: The Multilingual-Assisted Chat Interface flowchart

Interoperability is one of the most interesting features in our web service: we can use our web service to build new tools on any platform (other metaverse platforms, web services, applications, etc.). Note that it is used both for the Multilingual-Assisted Chat Interface in *Second life* and in *Solipsis*. Externalizing tools on PHP servers makes them platform-independent and saves a lot of time when applications must be adapted to different platforms.

Many web services exist on the web and propose specific ways to process information. Most of the time, there are separate tools for translating, for finding synonyms or for getting the definition of a word (Google Translate, WordNet, ConceptNet, etc.). Our web service uses all those tools in order to allow the user to centralize all this information in only one MLIF data structure.

4.5 Virtual Worlds Programming

The multilingual-assisted chat interface is now available in two virtual worlds: *Second Life*, developed by Linden Lab, and *Solipsis*, developed by Orange Labs and ArtefactO. This section describes our work in a technical way and is therefore directed to people who have an advanced knowledge of programming.

4.5.1 Snowglobe

Snowglobe is an alternative viewer for *Second Life* that can be downloaded on the official web site at http://snowglobeproject.org. This project aims at gathering Linden Lab and open source developers in order to create an innovative viewer.

The developers who want to implement new features can get the source code on the *Second Life* wiki (http://wiki.secondlife.com/). This page contains the source code, artwork and open source libraries for all operating systems. All the instructions concerning compilation and private linked libraries are detailed on the wiki (http://wiki.secondlife.com/wiki/Snowglobe) in the "Get source and compile" section.

We are now going to explain how to modify the chat interface and what we have modified in order to build the Multilingual-Assisted Chat Interface.

Modifying the chat interface. A user can send a message with two kinds of tools: the public chat interface and the instant messaging interface (i.e. private messages to one specific user).

- **The public chat interface:** The text written into the chat interface can be modified in two ways. The first one consists in modifying the text before sending it to the server and the second one consists in modifying the text upon its reception by the viewer.

  For the first case, the code is located in the files named *llchatbar.cpp*, *llchatbar.h* and *llchat.h*. The *LLChatBar* class is directly linked to the *LLPanel* class (defined in the files *llpanel.cpp* and *llpanel.h*) which is in charge of displaying the menu elements, among the chat elements. In parallel, the text is sent to the server from the function named *send_chat_from_viewer*. This function has three parameters. The string parameter *utf8_out_text* is the text that the user has written. The *type* parameter (of type *EChatType*) indicates

the range of the message. The range will depend on whether the user wants to whisper or to shout. It will be different too if the message concerns all the members of a region, the owner of an object or just a single user. The last parameter is the channel. There are public channels and private channels. Generally, we use private channels to communicate with objects.

The code for the second case is mainly defined in the *LLTextEditor* class (see *lltexteditor.cpp* and *lltexteditor.h*). Once the text has been written by the user, received by the *LLPanel* class and sent to the server by the *LLChatBar* class, the *LLTextEditor* class deals with the next step: it prepares the text for the class which is in charge of the display.

LLTextEditor has three important functions: *appendColoredText*, *appendStyledText* and *appendText*. The *appendColoredText* function defines the style of the text and calls the *appendStyledText* function. The latter applies the style to the text and finds out the HTML addresses to replace them by clickable links. If a link is detected, a new style is defined and the *appendText* function is called. When the code has completed the analysis of the HTML links, the *appendText* function is called a second time with the previous style. Each style change requires the *appendText* function to be called. In other words, *appendText* can consider only one style at the same time.

- **The instant message interface:** There is another way to communicate, which allows the user to have private communications with other users. This means of communication is called *instant message* and is defined in another file: *llimpanel.cpp*. The main function is called *addHistoryLine*. It can edit history and call functions of the *LLTextEditor* class in order to write into the instant message panel.

The Multilingual-Assisted Chat Interface. The functions we built for the Multilingual-Assisted Chat Interface are mainly defined in the *LLTextEditor* class.

When the viewer receives text, the *buildMLIF* function converts it into a MLIF data structure thanks to the web service, which provides the grammatical category for each word. Then, three functions (*parseMLIFAndReturnCategories*, *parseMLIFAndReturnLinks* and *parseMLIFAndReturnWords*) are called to parse the MLIF data structure and they return respectively each word, its grammatical category and the link to its information.

In the following, a "grammatically tagged link" defines a clickable HTTP link which embeds the grammatical category information of the word. A grammatically tagged link allows the viewer to recognize a word category and to apply it to the corresponding style and URL. It is composed of a word and its grammatical tag (tag://word). For example, if you write the verb "be" and wish to colour it, the viewer will create a "verb://be" grammatically tagged link. We chose to represent the links like this for implementation reasons. In fact, this was the clearest way to enable coloration and clickability of words in Snowglobe, since it is the same structure as the usual "http://website" links. The main format is MLIF; the "tag://word" is only a bridge between MLIF and the chat interface in Snowglobe.

The *addTags* function matches a word with its grammatical category in order to create a grammatically tagged link and to apply the colour chosen by the user. We have defined as many grammatically tagged link types as there are grammatical categories.

When user A wants an instant translation in their language of what user B says, the function *buildMLIF* requests the web service to return a MLIF data structure with user Bís sentence in its original language and the same sentence translated in user A's language. Both original and translated sentences are stored in the MLIF data structure. Indeed, user A can consult the original sentence if the translation is odd or if they want to learn the other language. The MLIF data is parsed by the*parseMLIFAndReturnTranslatedWords* function, which returns translated words.

All those functions are not called at the same time. A menu proposes several options (see 4.2). Each option is saved in an XML file (*panel_preferences_linguistic_chat.xml*), which is managed by the functions defined in the files *llprefslingchat.cpp*, *llfloaterpreference.cpp* and *llstartup.cpp*. The *appendStyledLinguisticText* function in the *LLTextEditor* class can verify if an option is activated before calling the associated function. For instance, if the variable *linguisticChatActivateInstantTranslation* is activated, the viewer will use the function which allows an instant translation of a sentence to be displayed(*parseMlifAndReturnTranslatedWords*).

4.5.2 Solipsis

Solipsis is a French virtual world (still under development), which is notable for two features. First, it plans to implement a decentralized Metaverse platform that will use a peer-to-peer protocol. Second, it integrates a web browser directly in the three-dimensional space. All the objects created can embed a web

browser (as a texture applied on a surface), which is compatible with many web technologies (Flash, Javascript, etc.).

The Multilingual-Assisted Chat Interface on Solipsis is roughly similar to the Snowglobe version. The general development is typically the same but the structure is different.

The graphical chat interface is defined in an HTML file (*uichat.html*) and built with HTML and Javascript. All the classes and functions we have described for the Snowglobe version are written in C++ in the files *GUI_Chat.cpp* and *GUI_Chat.h*.

The HTML file contains a form linked to a Javascript function (*sendMessage*). This calls a C++ function of *GUI_chat.cpp* (*addText*). This function decides, with respect to the user's preference of whether to use the linguistic functionalities or not, to convert the text to MLIF data or not. In other words, it calls one of the two sending methods declared in the HTML file:

1. *sendTextMessage*, which writes text into the chat interface without data processing;
2. *sendMLIFMessage*, which converts MLIF data to human-readable text (with the function *convertMlifToText*), and matches the tags with both corresponding colours and links (with the function *colorWordByTagAndMatchLink*).

The menu is implemented in the HTML file *menuLinguisticChat.html* and the actions that the user can execute with the in-world interface menu are declared with *showMenu* C++ function. When a user clicks on a word, a new in-world web browser appears thanks to the *showBrowser* C++ function.

5 CONCLUSION

First, we would like to highlight again that all the data exchanges are made using MLIF, because we intend to enhance interoperability through standardization. We believe standardization to be a key issue in the development and dissemination of new technologies.

Up to now, the grammatical coloration of the words in our Multilingual-Assisted Chat Interface interface is only available in English. In fact, such an analysis has to be based on linguistic corpora and the one we use only contains English terms. However, we plan to adapt our technology to other analysers such as Treetagger.

In the same way, the translations rely on the Google Translate web application. As its quality of translation is improving every day (thanks to its "suggest" functionality), we can expect better translations in the future.

The other important point which should be noted is that the web service which we developed has been used for both Second Life and Solipsis. Externalizing applications (especially service applications) is very useful in order to minimize the development time and to be able to use more adapted data structures.

But the Multilingual-Assisted Chat Interface is still a prototype. In the future, we intend to improve the reliability of this interface. We will distribute the modified Snowglobe viewer to a large number of users for testing and collect feedbacks. Adding new features to the chat interface in order to facilitate social interactions and e-learning is a process which still requires time, research and new technologies.

REFERENCES

Brill, E. (1992). A simple rule-based part of speech tagger. In *Proceedings of the third conference on Applied natural language processing*, pages 152–155. Association for Computational Linguistics.

Coleman, D. (2007). Flame First, Think Later: New Clues to E-Mail Misbehavior. *New York Times*.

Cruz-Lara, S., Bellalem, N., Bellalem, L., and Osswald, T. (2009). Immersive 3D Environments and Multilinguality: Some Non-Intrusive and Dynamic e-learning-oriented Scenarios based on Textual Information. *Journal of Virtual Worlds Research*, 2(3).

Cruz-Lara, S., Romary, L., Ducret, J., Bellalem, L., Guinaud, J., and Osswald, T. (n.d.). MLIF: the MultiLingual Information Framework. http://mlif.loria.fr/.

De Lucia, A., Francese, R., Passero, I., and Tortora, G. (2009). Development and evaluation of a virtual campus on Second Life: The case of SecondDMI. *Computers & Education*, 52:220–233.

Doughty, C. J. and Long, M. H. (2003). Optimal Pshycholinguistic Environments for Distance Foreign Language Learning. *Language Learning & Technology*, 7(3):50–80.

El Jed, M., Pallamin, N., and Pavard, B. (2006). Vers des communications situées en univers virtuel. In *National Conference on: Coopération, Innovation et Technologie*.

Suler, J. (2004). The Online Disinhibition Effect. *CyberPsychology and Behavior*, 7:321–326.