Online Generic Editing of Heterogeneous Dictionary Entries in Papillon Project

Mathieu MANGEOT
Unit Terjemahan Melalui Komputer
Universiti Sains Malaysia,
11800, Pulau Pinang
Malaysia
mathieu@mangeot.org

David THEVENIN
National Institute of Informatics
Hitotsubashi 2-1-2-1913 Chiyoda-ku
JP-101-8430 Tokyo
Japan
thevenin@nii.ac.jp

Abstract

The Papillon project is a collaborative project to establish a multilingual dictionary on the Web. This project started 4 years ago with French and Japanese. The partners are now also working on English, Chinese, Lao, Malay, Thai and Vietnamese. It aims to apply the LINUX cooperative construction paradigm to establish a broad-coverage multilingual dictionary. Users can contribute directly on the server by adding new data or correcting existing errors. Their contributions are stored in the user space until checked by a specialist before being fully integrated into the database. The resulting data is then publicly available and freely distributable. An essential condition for the success of the project is to find a handy solution for all the participants to be able to contribute online by editing dictionary entries.In this paper, we describe our solution for an online generic editor of dictionary entries based on the description of their structure.

1 Introduction

The Papillon Project (Sérasset and Mangeot, 2001) is a cooperative project for a multilingual dictionary on the Web with the following languages: English, Chinese, French, Japanese, Lao, Malay, Thai and Vietnamese. The dictionary structure makes it very simple to add a new language at any time. It aims to apply the LINUX construction paradigm to establish a multilingual usage dictionary with broad-coverage.

This project is based on the participation of voluntary contributors. In order to be really attractive, this project must imperatively find a convenient solution so that contributors can easily edit the dictionary entries. Since Papillon dictionary is available on a web server and the contributors are located all around the world, the obvious solution is to implement an editor available online. Unfortunately, the existing solutions (HTML forms, java applets) have important limitations. Thus, we propose an entirely generic solution that can adapt very easily not only the interfaces to the various entry structures needing to be edited but also to the user needs and competences.

Firstly, we outline the issue addressed in this paper; and draw up an overview of the existing methods for dictionary entry edition. A presentation of the chosen method follows detailing its integration in the Papillon server. Finally, we show an example of the online edition of a dictionary entry.

2 Addressed Issue and Requirements

In this paper, the addressed issue is how to edit online dictionary entries with heterogeneous structures.

2.1 Online Edition

In order to build a multilingual dictionary that covers a lot of languages, we need large competences in those languages. It may be possible to find an expert with enough knowledge of 3 or 4 languages but when that number reaches 10 languages (like now), it is almost impossible. Thus, we need contributors from all over the world.

Furthermore, in order to avoid pollution of the database, we plan a two-step integration of the contributions in the database. When a contributor finishes a new contribution, it is stored into his/her private user space until it is revised by a specialist and integrated into the database. Then, each data needs to be revised although the revisers may not work in the same place of the initial contributors.

Thus, the first requirement for the editor is to work online on the Web.

2.2 Heterogeneous Entry Structures

The Papillon platform is built for generic purposes. Thus, it can manipulate not only the

Papillon dictionary but also any kind of dictionary encoded in XML (Mangeot, 2002). The lexical data is organized in 3 layers:

- Limbo contains dictionaries in their original format and structure;
- Purgatory contains dictionaries in their original format but encoded in XML;
- Paradise contains the target dictionary, in our case Papillon dictionary.

The Purgatory data can be reused for building the Paradise dictionary.

We would like then to be able to edit different dictionaries structures from Paradise but also from Purgatory. Furthermore, being Papillon a research project, entry structures may evolve during the life of the project, since they are not fixed from the beginning.

Hence, the second requirement is that the editor must deal with heterogeneous and evolving entry structures.

2.3 Extra Requirements

Previous requirements must be fulfilled, whilst the following ones are optional.

The contributors will have various competences and use the editor for different purposes (a specialist in speech may add the pronunciation, a linguist may enter grammatical information, a translator would like to add interlingual links, and a reviewer will check the existing contributions, etc.).

The second optional requirement concerns the adaptation to the user platform. The increasing number of smart mobile phones and PDAs makes real the following scenarios: adding an interlingual link with a mobile phone, adding small parts of information with a PDA and revising the whole entry with a workstation.

It would then be very convenient if the editor could adapt itself both to the user and to the platform.

2.4 Final Aim

Guided by these requirements, our final aim is to generate, as much automatically as possible, online interfaces for editing dictionary entries. It has to be taken into account the fact that entry structures are heterogeneous and may vary and to try to adapt as much as possible these interfaces to the different kinds of users and platforms.

3 Overview of Existing Editing Methods

3.1 Local and Ad Hoc

The best way to implement a most comfortable editor for the users is to implement an ad-hoc application like the one developed for the NADIA-DEC project: DECID (Sérasset, 1997). It was conceived to edit entries for the ECD (Mel'čuk et al., 1984889296). The Papillon microstructure is based on a simplification of this structure. We were indeed very interested by such software. It is very convenient - for example - for editing complex lexical functions.

But several drawbacks made it impossible to use in our project. First, the editor was developed ad hoc for a particular entry structure. If we want to change that structure, we must reimplement changes in the editor.

Second, the editor is platform-dependent (here written and compiled for MacOs). The users have to work locally and cannot contribute online.

3.2 Distributed and Democratic

This solution implemented for the construction of the French-UNL dictionary (Sérasset and Mangeot, 1998) project is called "democratic" because it uses common and widespread applications (works on Windows and MacOs) such as Microsoft Word.

The first step is to prepare pre-existing data on the server (implemented here in Macintosh Common Lisp). Then, the data is converted into rtf by using a different Word style for each part of information (the style "headword" for the headword, the style "pos" for the part-of-speech, etc.) and exported. The clients can open the resulting rtf files locally with their Word and edit the entries. Finally, the Word rtf files are reintegrated into the database via a reverse conversion program.

This solution leads to the construction of 20,000 entries with 50,000 word senses. It was considered as a very convenient method, nevertheless, two important drawbacks prevented us to reuse this solution. The first is that in order to convert easily from the database to rtf and vice-versa, the dictionary entry structure cannot be too complex. Furthermore, when the user edits the entry with Word, it is very difficult to control the syntax of the entry, even if some Word macros can partially remedy this problem.

The second is the communication between the

users and the database. The Word files have to be sent to the users, for example via email. It introduces inevitably some delay. Furthermore, during the time when the file is stored on the user machine, no other user can edit the contents of the file. It was also observed that sometimes, users abandon their job and forget to send their files back to the server.

3.3 Online and HTML Forms

In order to work online, we should then use either HTML forms, or a Java applet. The use of HTML forms is interesting at a first glance, because the implementation is fast and all HTML browsers can use HTML forms.

On the other hand, the simplicity of the forms leads to important limitations. The only existing interactors are: buttons, textboxes, pop-up menus, and checkboxes.

JavaScripts offer the possibility to enrich the interactors by verifying for example the content of a textbox, etc. However, very often they raise compatibility problems and only some browsers can interpret them correctly. Thus, we will avoid them as much as possible.

One of the major drawbacks of this solution is our need to modify the source code of the HTML form each time we want to modify the entry structure. We also need to write as many HTML forms as there are different entry structures.

3.4 Online and Java Applets

In order to remedy the limitations of the HTML forms and to continue to work online, there is the possibility to use a java applet that will be executed on the client side. Theoretically, it is possible to develop an ad hoc editor for any complicated structure, like the 3.1 solution.

Nevertheless, the problems linked to the use of a java applet are numerous: the client machine must have java installed, and it must be the same java version of the applet. Furthermore, the execution is made on the client machine, which can be problematic for not very powerful machines. Moreover, nowadays there is a strong decrease of java applets usage on the Web mainly due to the previous compatibility problems.

3.5 Conclusion

As a result, none of these existing solutions can fully fulfil our requirements: online edition and heterogeneous entry structures. We might then use other approaches that are more generic like the ones used in interface conception in order to build our editor. In the remainder of this paper, we will detail how we used an interface generation module in Papillon server in order to generate semi-automatically editing interfaces.

4 Using an Interface Generation Module

This Papillon module has to generate graphic user interfaces for consulting and editing dictionary entries. We base our approach on the work done on Plasticity of User interfaces (Thevenin and Coutaz, 1999) and the tool ART-Studio (Calvary et al., 2001). They propose frameworks and mechanisms to generate semi-automatically graphic user interfaces for different targets. Below we present the design framework and models used.

4.1 Framework for the UI generation

Our approach (Calvary et al., 2002) is based on four-generation steps (Figure 1). The first is a manual design for producing initial models. It includes the application description with the data, tasks and instances models, and the description of the context of use. This latter generally includes the platform where the interaction is done, the user who interacts and the environment where the user is. In our case we do not describe the environment, since it is too difficult and not really pertinent for Papillon. From there, we are able to generate the Abstract User Interface (AUI). This is a platform independent UI. It represents the basic structure of the dialogue between a user and a computer. In the third step, we generate the Concrete User Interface (CUI) based on the Abstract User Interface (AUI). It is an instantiation of the AUI for a given platform. Once the interactor (widget) and the navigation in UI have been chosen, it is a prototype of the executable UI. The last stage is the generation of Final User Interface (FUI). This is the same as concrete user interface (CUI) but it can be executed.

We will now focus on some models that describe the application.

4.2 Application Models: Data & Task

The Data model describes the concepts that the user manipulates in any context of use. When considering plasticity issues, the data model should cover all usage contexts, envisioned for the interactive system. By doing so, designers obtain a global reusable reference model that can be specialized according to user needs or

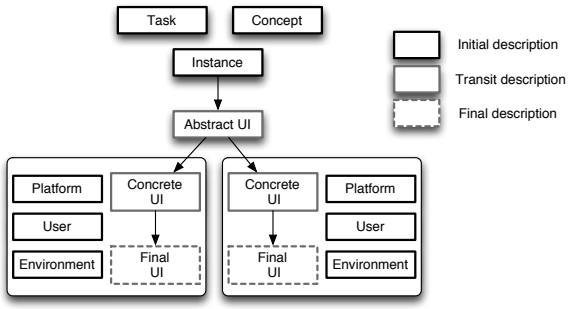

Figure 1: Multitarget Generation Framework

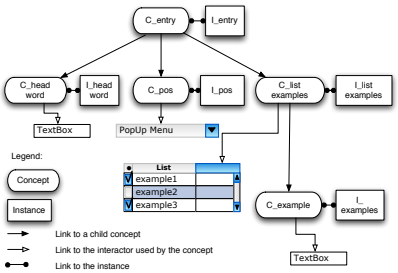

Figure 2: Data Model Structure

more generally to context of use. A similar design rationale holds for tasks modeling. For the Papillon project, the description of data model corresponds to the XML Schema description of dictionary and request manipulation. The tasks' model is the set of all tasks that will be implemented independently of the type of user. It includes modification of the lexical database and visualization of dictionaries.

As showed on Figure 2, the model of concepts will drive the choice of interactors and the structure of the interface.

4.3 Instance Model

It describes instances of the concepts manipulated by the user interface and the dependence graph between them. For example there is the concept "Entry" and one of its instances "scientifique". (cf. Figure 3).

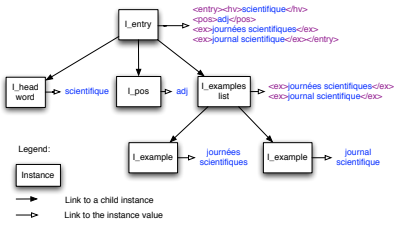

Figure 3: Relation Between the XML Entry and its Corresponding Instance

This model is described at design time, before generation, and linked with the task model (a task uses a set of instances). Each instance will be effectively created at run-time with data coming from the Papillon database.

4.4 Platform and Interactors Models

A platform is described by interaction capacity (for example, screen size, mouse or pen, keyboard, speech recognition, etc.). These capacities will influence the choice of interactors, presentation layouts or the navigation in the user interface.

Associated to the platform there are the interactors (widgets) proposed by the graphic toolsbox of the targeted language (for example Swing or AWT for Java). In this project interactors are coming from HMTL Forms (textBox, comboBox, popup menu, button, checkBox, radioButton) and HTML tags. We also had to build more complex interactors by a combination of HTML Forms and HTML Tags.

4.5 User Model

Previous research has shown the difficulty to describe the cognitive aspects of user behavior. Therefore, we will simplify by defining different user classes (tourist, student, business man, etc.). Each class will be consisting of a set of design preferences. Depending on the target class, the generator will use appropriate design rules.

The model is not yet implemented; it is implicitly used in the data & task models. We defined different views of data according to the target:

- all data is rendered for the workstation editing interface for lexicographers,
- only headword and grammatical class are rendered and examples are browsable on the mobile phone interface for a "normal" dictionary user.

4.6 Concrete User Interface Model

This model, based on an independent user interface language, describes the graphic user interface, as the final device will render it. It is target-dependent.

4.7 Final User Interface

From the CUI model, the generator produces a final interface that will be executed by the targeted device, and links it with the Papillon database. In our case we produce:

- HTML code for the workstation,

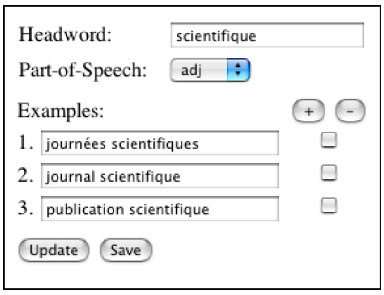

Figure 4: Generated GUI

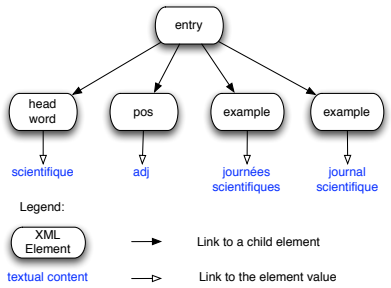

Figure 5: Abstract View of an Entry

- Tiny XHTML code for AU mobile phones,
- and CGI links for the communication with the database.

Figure 4 shows a simple example of a final generated UI.

5 Integrating the Module in Papillon Server

5.1 Implementation

The Papillon server is based on Enhydra, a web server of Java dynamic objects. The data is stored as XML objects into an SQL database: PostgresQL.

ARTStudio tool is entirely written in Java. For its integration into the Papillon/Enhydra server, we created a java archive for the codes to stay independent.

The Papillon/Enhydra server can store java objects during a user session. When the user connects to the Papillon server with a browser, a session is created and the user is identified thanks to a cookie. When the user opens the dictionary entry editor, the java objects needed for the editor will be kept until the end of the session.

5.2 A Working Session

When the editor is launched, the models corresponding to the entry structure are loaded. Then, if an entry is given as a parameter (editing an existing entry), the entry template is instantiated with the data contained in that entry. If no entry is given, the template is instantiated with an empty entry. Finally, the instantiated models and entry templates are stored into the session data and the result is displayed embedded in an HTML form, through a Web page (Figure 4).

Then, after a user modification (e.g. adding an item to the examples list), the HTML form sends the data to the server via a CGI mechanism. The server updates the models and template stored in the session data and sends back the modified result in the HTML page.

At the end of the session, the modified entry is extracted from the session data and then stored as a contribution in the database.

6 An Editing Example

6.1 A Dictionary Entry

Figure 5 shows an abstract view of a simple dictionary entry. It is the entry "scientifique" (scientific) of a French monolingual dictionary. The entry has been simplified on purpose. The entries are stored as XML text into the database.

6.2 Entry Structure

The generation of the graphic interface is mostly based on the dictionary microstructure. In the Papillon project, we describe them with XML schemata. We chose XML schemata instead of DTDs because they allow for a more precise description of the data structure and handled types. For example, it is possible to describe the textual content of an XML element as a closed value list. In this example, the French part-of-speech type is a closed list of "nom", "verb", and "adj".

Figure 6 is an abstract view of the structure corresponding to the previous French monolingual dictionary entry.

6.3 Entry Displayed in the Editor

The dictionary entry of Figure 5 is displayed in the HTML editor as in Figure 4. In the following one (Figure 7), an example has been added in the list by pushing the + button.

6.4 A More Complex Entry

In the following figure (Figure 8), we show the entry 食べる (taberu, to eat) of the Papillon Japanese monolingual volume. The entry structure comes from the DiCo structure (Polguère,

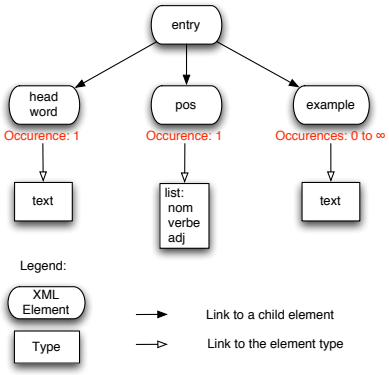

Figure 6: Structure of an Entry

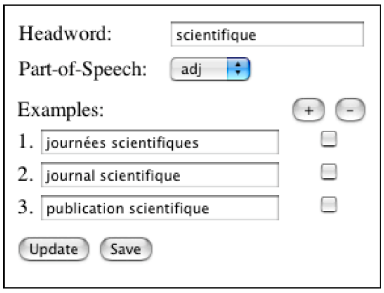

Figure 7: Entry Displayed in the Editor

2000), a light simplification of the ECD by Mel'čuk & al.

Two interesting points may be highlighted. You can note that not only the content of the entry is in Japanese, but also the text labels of the information. For example, the first one, 見出し語 (midashigo) means headword. The interface generator is multitarget: it generates the whole HTML content. It is then possible to redefine the labels for each language.

The second point is the complexity of the entry structure. There is a list of lexical functions. Each lexical function consists of a name and a list of valgroups (group of values), and in turn, each valgroup consists of a list of values. Finally, each value is a textbox. The lists are nested the one in the other one and it is possible to use the lists + and - operators at any level.

7 Evaluation

7.1 Preamble

This paper focuses on one particular functionality of the Papillon platform: the generic editor. Its purpose is not to present the building of Papillon dictionary or the progress of Papillon Project as a whole. The evaluation will then focus on the editor.

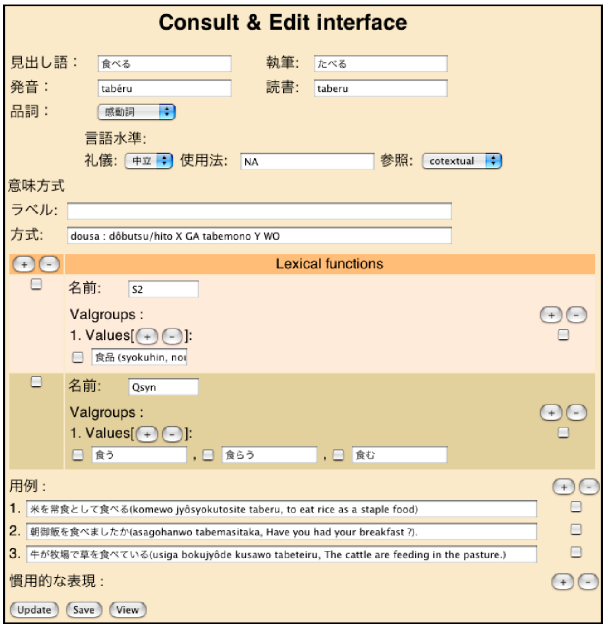

Figure 8: Papillon Entry Displayed in the Editor

The contribution phase on Papillon project has not begun yet. Thus, for the moment, very few users tested the editor. We have not yet enough data to evaluate seriously the usability of the interface. Then, the evaluation will be driven on the technical aspects of the editor.

In order to evaluate the genericity and the usability of the editor, we generated interfaces for two other dictionary structures: the GDEF Estonian-French dictionary and the WaDokuJiTen Japanese-German dictionary.

7.2 Edition of the GDEF dictionary

The GDEF project (Big Estonian-French Dictionary) is managed by Antoine Chalvin from INALCO, Paris. The dictionary microstructure is radically different from the Papillon dictionary as you will see in Figure 9 compared to figure 8. You may notice the 6 levels of recursion embedded in the entry structure.

It took about one week to write the interface description files for the new dictionary structure in order to generate properly a complete interface for the GDEF dictionary.

7.3 Edition of the WaDokuJiTen

The WaDokuJiTen project is managed by Ulrich Apel, now invited researcher at NII, Tokyo. The dictionary is originally stored in a FileMaker database. It has more than 200,000 entries. It took four days to export integrate the dictionary in Papillon platform and to write the

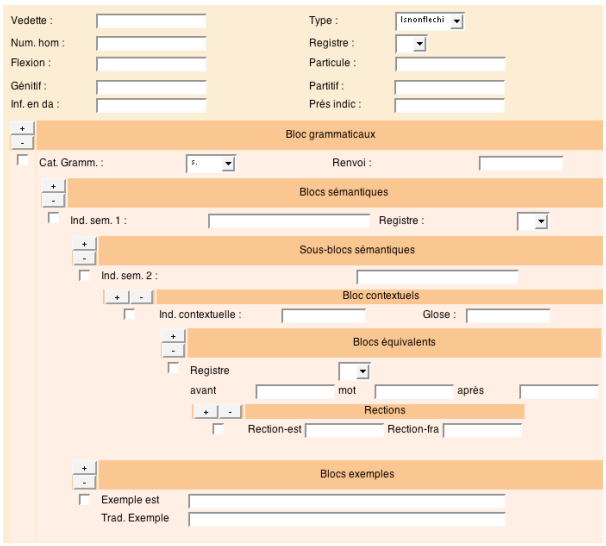

Figure 9: GDEF Entry Displayed in the Editor

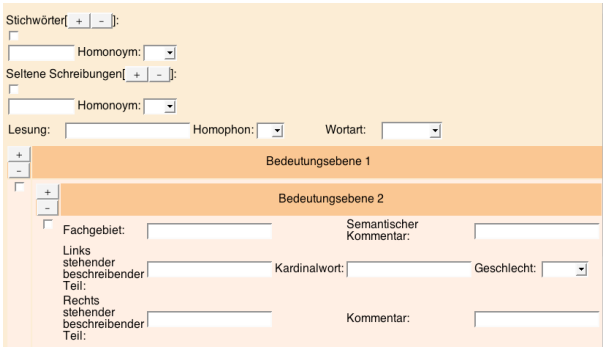

Figure 10: WaDokuJiTen Entry Displayed in the Editor

files needed for the generation of the editor interface. The integration was done in 4 steps: export the dictionary from FileMaker into an XML file, tag the implicit structure with a perl script, write the metadata files and upload the dictionary on the Papillon server.

The dictionary microstructure is simpler than the previous one (see figure 10). It took only two days to write the files needed for the generation of the editor interface.

8 Conclusion

The implementation of ARTStudio and Papillon plateform started separately four years ago. The development of the HTML generation module in ARTStudio and its integration into Papillon platform took about a year from the first specifications and the installation of a complete and functional version on the Papillon server. The collaboration between a specialist of computational lexicography and a specialist of the adaptability of interfaces has produced very original and interesting work. Furthermore, the evaluation and the feedback received from the users is very positive. Now, we want to further pursue this work following several paths.

First of all, only a specialist can use the existing interface for Papillon entry since it is too complex for a beginner. We plan to generate different interface types adapted to the varied user needs and competences. Thanks to the modularity of the editor, we need only to describe the tasks and instance models corresponding to the desired interface.

For the moment, the interface generation is not fully automatic; some of the model descriptions used by the editor have to be written "by hand". This is why we are working now on automating the whole generation process and the implementation of graphical editors allowing users to post-edit or modify a generated interface description.

References

Gaëlle Calvary, Joëlle Coutaz, and David Thevenin. 2001. Unifying reference framework for the development of plastic user interfaces. In *EHCI'01, IFIP WG2.7 (13.2) Working Conference*, pages 173–192, Toronto, Canada.

Gaëlle Calvary, Joëlle Coutaz, David Thevenin, Quentin Limbourg, Nathalie Souchon, Laurent Bouillon, and Jean Vanderdonckt. 2002. Plasticity of user interfaces: A revised reference framework. In *Proc. TAMODIA 2002*, pages 127–134, Bucharest, Romania. INFOREC Publishing House.

Mathieu Mangeot. 2002. An xml markup language framework for lexical databases environments: the dictionary markup language. In *International Standards of Terminology and Language Resources Management*, pages 37–44, Las Palmas, Spain, May.

Igor Mel'čuk, Nadia Arbatchewsky-Jumarie, Léo Eltnisky, Lidija Iordanskaja, Adèle Lessard, Suzanne Mantha, and Alain Polguère. 1984,88,92,96. *DEC : Dictionnaire explicatif et combinatoire du français contemporain, recherches lexico-sémantiques I,II,III et IV*. Presses de l'université de Montréal, Canada.

Alain Polguère. 2000. Towards a theoretically-motivated general public dictionary of semantic derivations and collocations for french. In *Proceeding of EURALEX'2000, Stuttgart*, pages 517–527.

Gilles Sérasset and Mathieu Mangeot. 1998. L'édition lexicographique dans un système générique de gestion de bases lexicales multilingues. In *NLP-IA*, volume 1, pages 110–116, Moncton, Canada.

Gilles Sérasset and Mathieu Mangeot. 2001. Papillon lexical database project: Monolingual dictionaries and interlingual links. In *NLPRS-2001*, pages 119–125, Tokyo, 27-30 November.

Gilles Sérasset. 1997. Le projet nadia-dec : vers un dictionnaire explicatif et combinatoire informatisé ? In *LTT'97*, volume 1, pages 149–160, Tunis, Septembre. Actualité scientifique, AUPELF-UREF.

David Thevenin and Joëlle Coutaz. 1999. Plasticity of user interfaces: Framework and research agenda. In *Interact'99 Seventh IFIP Conference on Human-Computer Interaction*, volume 1, pages 110–117, Edinburgh, Scotland.