Títol: Desenvolupament i integració d'un glossari de termes i registre de sinònims per a ser usat en les eines del sistema PABRE.

Volum: 1/1
Alumne: Jonathan Cubel Figueroa

Director: Carme Quer Bosor
Departament: Enginyeria de Serveis i Sistemes d'Informació (ESSI)
Data: Gener 2014

DADES DEL PROJECTE

Títol del Projecte: ***Desenvolupament*** **i integració d'un glossari de termes i registre de sinònims per a ser usat en les eines del sistema PABRE.**

Nom de l'estudiant: **Jonathan Cubel Figueroa**

Titulació: **Enginyeria Informàtica**

Crèdits: **37,5**

Director: **Carme Quer Bosor**

Departament: **Enginyeria del Software i Sistemes d'Informació (ESSI)**

MEMBRES DEL TRIBUNAL *(nom i signatura)*

President: **Xavier Franch Gutiérrez**

Vocal: **Julita Corbalán González**

Secretari: **Carme Quer Bosor**

QUALIFICACIÓ

Qualificació numèrica:

Qualificació descriptiva:

Data:

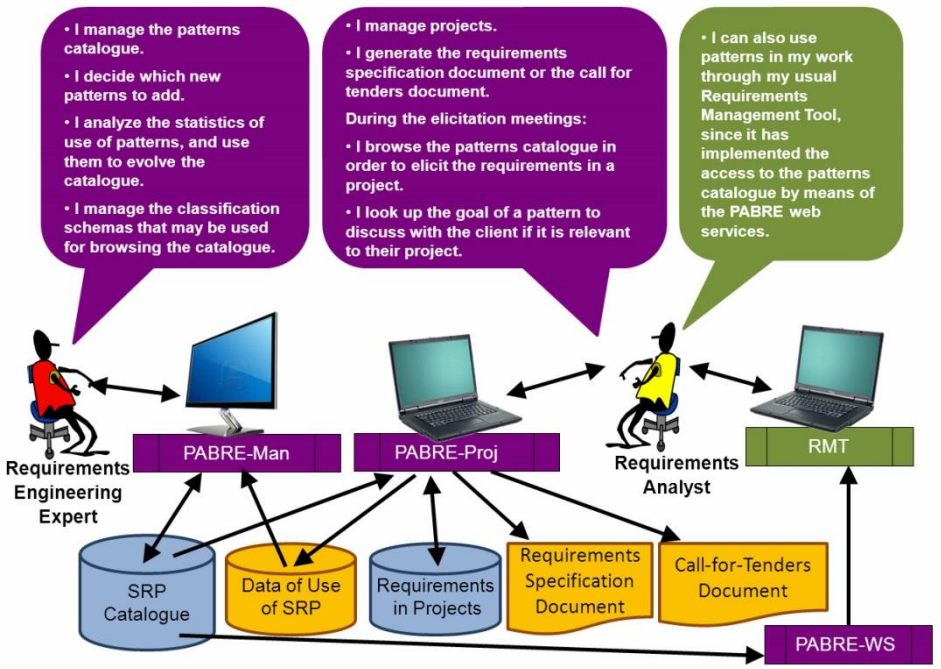
Content

1 Introduction

The reuse of Software Requirements, as a process of Software Engineering, gives the chance to Requirement Engineers to maximize the quality of contents and syntax of Software Requirement Specifications during the elicitation, validation and documentation phase. Addressing this quality maximization, Requirement Patterns play an important role upon this quality maximization regarding reusability, diminishing time and error factors whilst defining Software Requirement Specifications (SRS).

The GESSI Research Group of the Universitat Politècnica de Catalunya Barcelona Tech, with the help of SSI Department of Public Research Centre Henri Tudor (TUDOR, Luxembourg) have built a first version of a catalog of Software Requirements Patterns (SRP) composed of a set of functional, non-functional and non-technical patterns. These patterns can be used and managed by means of the Pattern-Based Requirements Elicitation system (PABRE), also developed by GESSI.

The PABRE system (see figure below) is composed of three tools: PABRE-Man, PABRE-Proj and PABRE-WS. PABRE-Man is the tool within the PABRE system that helps to construct, and make evolve SRP. PABRE-Proj is the tool that supports the phase of elicitation of requirements of a certain project and provides functionalities for the generation of SRS and other documents as Call-for-Tenders. The SRP are stored in the SRP Catalogue which is accessed by both tools. The system also provides PABRE-WS, which are web services that allow accessing to the SRP Catalogue to the Requirement Management Tools (RMT) existent in the marketplace. [1]



The PABRE system

Without going into detail (for more information see [2]), each SRP is structured in two parts: the SRP general attributes and one or more SRP Forms. Both the SRP and its forms have *Name*, a *Description*, an *Author*, *Comments* and *Sources* sections. The SRP has also the *Goal* section, and the *Dependencies* section. Each form also consists on several sections the most important of which are one *Fixed Part* and several *Extended Parts*. These parts have the same attributes: *Question Text*, *Form Text* both having *Parameters* in the text. For each parameter a metrics is defined. The following example shows the structure of a SRP:

Requirement Pattern *Authentication*	**Description**	This pattern expresses the need of having the system functionality to identify users	
	Comments	----	
	Goal	Ensure the identity of the users that access to the system.	
	Author	GESSI-CITI	
	Sources (0..*)	• Requirement books from CITI • Specialized literature	
	Keywords (0..*)	Access Control, Security, Users	
	Dependencies (0..*)	IMPLIES Authorization IMPLIES Stored Data Protection	
Requirement Form *Authentication*	**Description**	This form states the general need of having the system functionality of identifying users, and has extensions for detailing the type or technology to be used. An extension for requiring to not be necessary to create an specific account for the system is present.	
	Comments	Application of extensions: Authentication Technology, Single Sign-on: may be applied at most once each.	
	Version date	2009-03-20 00:00:00.0	
	Author	GESSI-CITI	
	Sources (0..*)	• Requirement books from CITI • Specialized literature	
	Fixed Part	**Question text**	----
		Form text	The system shall authenticate users
	Extended Part *Authentication Technology*	**Question text**	----
		Form text	The authentication process shall be based on the %authMechanism% authentication technology
		Parameter	**Metric**
		authMechanism: is an authentication software technology	AuthenticationTechnology: AuthenticationTechnology = Domain(Windows login, …)
	Extended Part *Authentication Types*	**Question text**	----
		Form text	The authentication type shall be: %authTypes%
		Parameter	**Metric**
		authTypes: is a non-empty set of authentication types	AuthenticationTypes: AuthenticationTypes = Set(AuthenticationType) AuthenticationType = Domain(Open, Encrypted)
	Extended Part *Single Sign-on*	**Question text**	----
		Form text	The system shall not oblige users to create and manage a specific account for this system

Authentication SRP Structure

A Requirement Engineering Expert, from now on User, uses the PABRE-Proj tool to introduce new SRP in the SRP Catalogue. He or she has to provide a value for the attributes of the SRP enumerated before throw the fields and forms of the PABRE-Proj interface. Aside of some fields for which specific constraints have to be fulfilled as in the case of date fields (for instance in the case of the Version Date field), there are other textual fields that have no restrictions neither in semantics and quantity of the written text (for instance Question Text field, Description field or Form Text field). Regarding this last situation, a Requirement Engineering Expert may write whatever text considered being proper to fulfill the specifications of the given field. Perhaps, due to different contexts, phraseology, word choice, maybe the same significant, concept or artifact is referred to with different expressions. To avoid this situation, which is known as Hyponymy, a mechanism should be implemented and added to the PABRE-Man system. This project consists on the development the mechanism which enables hyponymy substitution in order to achieve the objective of finding the most general and accepted word (or the word selected as the more general for the Requirement Engineering Experts that manage a SRP Catalogue) in SRP definitions

The concept of hyponymy has a direct application to this system since its main goal is the substitution of specific terms from one general term which includes its meaning. In linguistics, a hyponym is a word whose meaning is included within that of another word, its hypernym. For example, Beagle, Mastiff, Alaskan malamute, and Rottweiler are all hyponyms of dog (their hypernym), which is, in turn, a hyponym of animal.

In computer science, especially in the field of Artificial Intelligence, when ontologies are defined, often the relationship is named an "is-a" relationship. This is why the semantic relationship between a hyponym and its hypernym answers to the "is-a" (should not be confused with the "part-of" relationship) relation as in, for instance, "A Rottweiler 'is-a' dog".

Hypernymy is the inverse semantic relation in which one word is the hypernym of another, for instance, dog is hypernym of Mastiff. In other words, hypernymy corresponds to the relation of class to subclass. Resuming, a hypernym can be understood as a more general word than its hyponym, denotes the relation that is used the same way as generalization vs. specialization. [3]

In this project the system must accept every possible word, phrase or expression to be held up as possible hyponym, regardless punctuation marks, and considering French, English and Spanish character sets and languages. These hyponyms will be identified in the main open text form fields of the PABRE-Man system, and its hypernym will be proposed to the user to improve the wording of the fields' content. For instance in the Form Text and Question Text fields.

2 Project's Goal

The main goal of this project is to extend the PABRE-Man tool with a mechanism which enables hyponymy substitution in order to achieve the objective of finding the most general and accepted word according to standards in SRP definitions - either when an SRP is created or edited for modification. Even though, the system shall give the chance to do not apply the substitution, the user will be the one that decides if this substitution will be carried out or not; in other words, the substitution shall not be automatic, it shall be decided by the user whether or not to apply the substitution. Anyway, the hyponyms or invalid words will be underlined, so the user knows that there is one hypernym or word more suitable for the SRP definition, to take his/her decision.

The project includes the study of alternative implementations of the glossary or dictionary that maintains the hyponyms and hypernyms relationships (from now on these will be the terms by means this document will refer to the set of relationships), the development of the glossary management by means of a feature added to PABRE-Man, and the integration of the glossary in the creation and edition of SRP's through PABRE-Man.

As a summary, the sub-objectives are the following:

- **Integration**: The new features must be integrated into PABRE-Man tool without any collateral damage or interaction.
- **Morphism application**: The new features shall make possible the hyponym substitution with respective hypernym.
- **Usability:** The user interface must be a user friendly and efficient, and specially ensure user can have incorrect words replaced in a fast way and with a non-tedious mechanism.
- **Hyponym detection**: The hyponyms must be detected runtime while the user is writing or pasting content to the PABRE-Man form fields.
- **Adding/Editing/Deleting glossary entries**: There must be a process in which the user can interact and manage the glossary.
- **Glossary persistence**: All changes in the glossary must be saved. User also may have the opportunity to discard changes.

In addition, the three-layer architectural design must be respected as it is throughout the PABRE system. This design derives the action responsibility to controllers which supervise presentation (look and feel, user interfaces), domain (business processes) and data (persistence). For this reason the new features must be adapted to the object-relational mapping already present in the PABRE-Man project relating with Hibernate library.

It is also noticeable that for every hypernym there has to be a description associated to it, but it could be void or null.

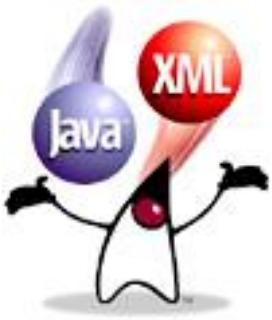
3 Technologies

During the elaboration of this project, several technologies have been used to arrive to a final satisfactory solution. Some of them where handicapped and pinned because of the architecture and previous development of the PABRE-Man tool and others where selected in consonance with the needs and future expansion of the same tool. Amongst them, these are the most significant:

The coding language of the PABRE tool is Java. Java has been tested, extended, and proven by a community of more than 9 million developers, architects and supporters worldwide. This means documentation is abundant for any aspect within the application. Java is also a good choice because software can be written on one platform and run on virtually any other. This is really important and useful when executing the application or deploying new versions. Java is also a good language to write with when searching Object Oriented Architectures or Three Layer Architectures amidst with REST-API in Web based Applications. Currently owned by Oracle, Java has a powerful public API, consistently with good documentation on-line forums and tutorials. A very powerful tool along with the wide number of developers, supporters and enterprises is the library creation and easy incorporation inside a workbench. The Java version used in the project is JavaSE-1.6. [4]

Hibernate is a clear example of the mentioned above. It is an object-relational mapping (ORM) library for the Java language, providing a framework for mapping an Object-oriented domain model to a relational database (SQL). With a simple instruction surrounded with a proper context- connection to Hibernate session, error and Exception handling- Hibernate generates SQL instructions and frees the developer from manual result set handling and object conversion. Besides, Hibernate is free software and is distributed under the GNU-GPL which tidies pretty much things up. [5]

Guava Google core libraries for Java 1.5+ Guava introduces a number of new collection types that are not in the standard *Java Development Kit* (JDK). As a general rule, the Guava collection implementations follow JDK interface contracts very precisely. Specifically the collection to use in this project is Multimap. Multimap is a collection similar to a Map, but which may associate multiple values with a single key. If you call put (K, V) twice, with the same key but different values, the Multimap contains mappings from the key to both values. It results to be excellent when fetching all the hyponyms of a hypernym so they can all be used within the presentation layer. Does not penalize efficiency or overlay contracts because insertion can be done at the same time and in the same manner as it is on normal Map. [6]



Simple API for XML (SAX) provides mechanisms for reading data from an XML document sequentially. It basically works upon certain event handlers that fire at the time of finding an XML tag when parsing the XML document sequentially. These handlers are:

- *startElement* (String namespaceURI, String localName, String qualifiedName, Attributes atts)

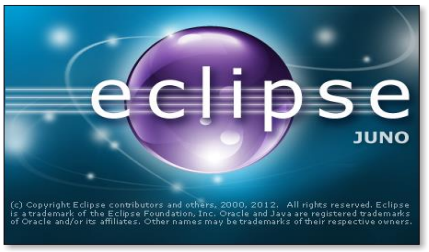
Describes the actions to perform when finding the starting element of an XML tag. The tag element content is located in the qualifiedName parameter.

- *Characters* (char[] ch, int start, int length)

Describes the action to be done while reading characters indexed by the parameters. Typically a local String variable is created using these indexes and later processed as desired.

- *endElement* (String Uri, String localName, String qualifiedName)

Describes the actions to perform when finding the ending element of an XML tag. The tag element content is located in the qualifiedName parameter. Generally centers the main actions to perform as it is clear an element has been parsed. [7]



Eclipse is an IDE for Java development created on November 2001 by means of the Eclipse Project which was composed by Industry leaders Borland, IBM, MERANT, QNX Software Systems, Rational Software, Red Hat, SuSE, TogetherSoft and Webgain. Since 2004 the Eclipse Foundation, as an independent not-for-profit Corporation acting as Steward for the Eclipse community, holds its development. Eclipse is undoubtedly one of the best tools for developing Java based software. It supports Java native applications in back-end and front-end systems, in Web Development or recently developing Android Applications. Eclipse also counts with a set of plugins (called Market) which make development easier in such a sense. The version and build in which this Project has been realized is the following:

Eclipse Java EE IDE for Web Developers. Version: Juno Service Release 2 Build id: 20130225-0426

Eclipse is distributed with EPL (Eclipse Public License), which allows its use when developing the PABRE application. [8]

SQuirreL SQL Client is a graphical Java program that allows viewing the structure of a JDBC compatible database, browsing the data in tables or executing SQL commands. Among its most suitable characteristics the following stand up:

- Easily view and edit data from any relational database. Using the UI, data can be even modified without writing a single SQL statement.
- View database metadata in a simple way, without SQL statements.
- Working with multiple databases, both local and remote machines.
- Inline execution of SQL statements to modify the database or information retrieval.

In the development of the project SQuirreL SQL has been used to create the persistence Schema of the Glossary relations in the Derby database and to verify Hibernate functionality and correctness of the data inserted. However SQuirreL SQL client works with any Database Management System which works with SQL and has an available JDBC driver. [9]

4 Requirements

System requirements are often defined as “A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document”. It is thereby proper to define accurate requirements in a project dealing with them.

4.1 Functional Requirements

As stated before, the functionalities that the project has to provide are the following:

- Implementation of the glossary
- Development of the glossary management
- Integration of the glossary in the creation and edition of SRP’s through PABRE-Man.

Implementation of the Glossary

The glossary is the dictionary that maintains the hyponyms and hypernyms relationships. To implement this glossary several alternatives for the design and implementation of the glossary will be considered and the more suitable to the PABRE system will be chosen.

Development of the glossary management

A new functionality will be added to PABRE-Man in order to facilitate:

- Add new hypernyms
- Change the name or description of a hypernym
- Add hyponyms to hypernyms.

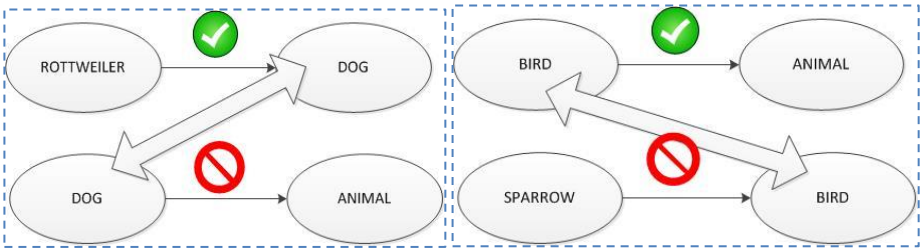
The new feature will have to be well integrated with PABRE-Man and have the same look and feel than the other functionalities in this tool.

The relationships among the hypernyms and hyponyms will have to follow some integrity constraints that I outline in the following paragraphs. For visualization purposes and a better understanding of these relationships, I use graphical representation of the relationships and the integrity constraints they have to fulfill.

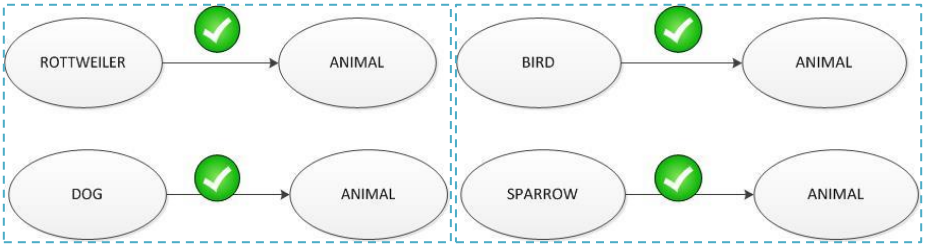
The glossary relations can be transposed to a mathematical model using nodes representing Terms and directed edges representing relationships among Terms. Edges are incident from a hyponym node to a hypernym node. Thus, the system must fulfill the following functional requirements:

1. **One level hierarchy:** there must be a situation where a hyponym is hypernym of another hyponym or a hypernym is hyponym of another hypernym.

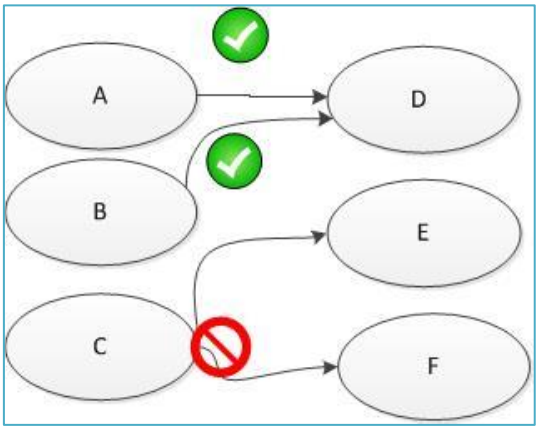
Graphically:



The solution to this problem relies on the semantics over the words themselves. A more general hypernym must be chosen to be de dominant hypernym, as follows:



2. **No Graph**: One hyponym can only be hyponym of one, and only one, hypernym. However hypernyms can have more than one hyponym (semantically it should even be a quality measure for a hypernym to have great number of hyponyms).



It is also important how these requirements are going to be fulfilled. Every time a hypernym or hyponym is added or modified, these conditions must be checked in order for the requirements to be met. More to this will be added in the design chapter of this document.

Integration of the glossary in PABRE-Man

As I stated before, the use of the glossary will be during the creation or edition of a SRP.

- The hyponyms shall be identified and this situation will be showed to the user by underlining them.
- When a hyponym shall be selected by the user and the user shall press the right button of the mouse, the hypernym of the hyponym will be proposed to the user.
- The user will choose if the hyponym is changed by the hypernym or not.

This functionality will be introduced to the following form fields of the edition of SRP:

- Name
- Description
- Attributes
- Keywords
- Fixed Part Text of SRP Forms
    - Pattern Text
    - Question Text
- Extended Part Text of SRP Forms
    - Pattern Text
    - Question Text

It is important to take into account that this functionality would be easily extensible to other form fields of the PABRE-Man interface. In the case of the Pattern Text and Question Text form fields the analysis of hyponyms has to take into account that in the text there will appear special words named "parameters". These parameters are used during the application of the pattern, taking values specific of the project where the pattern is applied. These parameters are market in the text because they are words that begin and end with a character "%". The analysis of hyponyms does not have to be applied to these parameters. Thus, the parameters may be formally defined by:

$$\forall\, \omega \in \Sigma^* \,/\quad \omega = \text{'\%'}\mu\text{'\%'} \;\rightarrow\; \mu \in \text{Parameter}$$

Upon parameters there shall be no hyponym analysis, due to the special kind of such word. Moreover, within the word containing the parameter-like structure there must be no whitespaces, although hyphens are allowed. Thus, the phrase:

- “American Mastiff %Mastiff% and Abyssinian are the most common pets in Imaginary Ville”

Supposing there is a hyponym entry having Abyssinian and cat, after applying hyponym substitution will change to:

- “Dog %Mastiff% and Cat are the most common pets in Imaginary Ville”.

Notice that substitution mimes the case of the substituted word.

4.2 Non-Functional Requirements

The non-functional requirements of this project are the ones that guarantee the maximization of several non-functional factors. The non-functional factors of the quality model proposed by the ISO/IEC 25010 can be used.

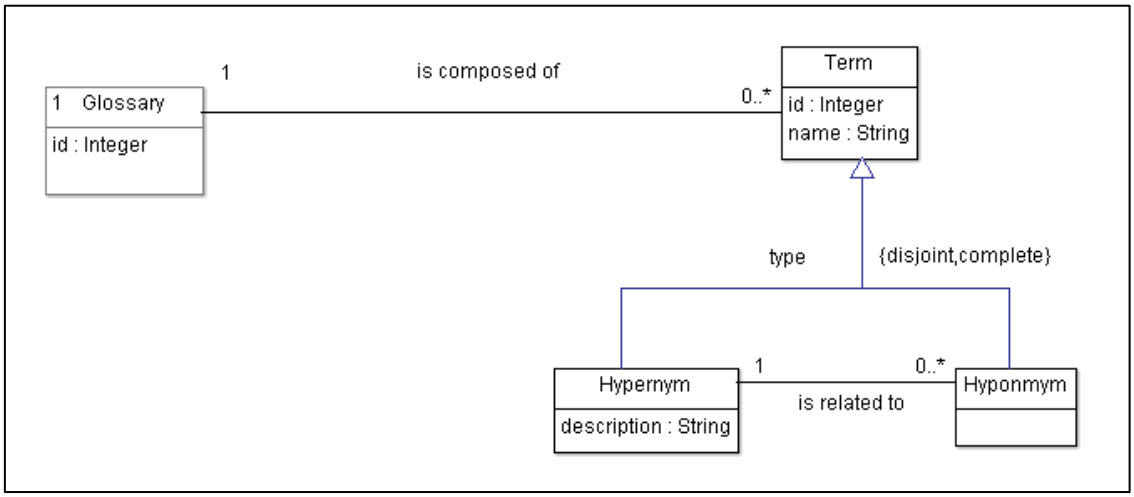
This quality model categorizes software product quality properties into eight characteristics: functionality (requirements already described in the previous section), efficiency, compatibility, usability, reliability, security, maintainability and portability. Each characteristic is composed of a set of related sub characteristics. Here I state for each non-functional characteristic, how the project has to afford it.

- **Efficiency** – In this project, the new features, and specifically the identification of hyponyms and their hypernym must be quick just after the writing of the words of the SRP attributes.
- **Compatibility** – The added software must co-exist and be interoperable with previous states of software and with the rest of the applications' components.
- **Usability –** The use of the glossary from the form of edition of SRP should be intuitive and easy to use. The idea is to use the typical interaction used in word orthographic correctors.
- **Reliability** – The project relies in the implementation of PABRE-Man for this characteristic.
- **Security**- The project relies in the implementation of PABRE-Man for this characteristic.
- **Maintainability** – In this project, the maintainability will be achieved if the architectural design of the PABRE-Man system that will be extended is respected, and the design patterns used in the system also used in its extension.
- **Portability** – The PABRE-Man for the technologies used in its development is portable to any system that supports Java, in this project the extension must use the same technologies or technologies that also respect this portability. For instance, as PABRE-Man is used in different platforms using the JVM, and some features are used in Web Services, the Glossary must also be portable to these systems by the means of the Java technology.

5 Specification

5.1 Conceptual Model

According to the requirements above and considering the constraints identified in the introduction, in this section the Conceptual Model of the new features that this project develops is presented:



External Keys: Glossary (id); Term (name)

This conceptual model has also to follow the integrity constraints stated in the functional requirements, which are: do not allow graphs and so, just allow one level of hierarchy. This is provided by the conceptual model because of the definition of the specialization as disjoint, complete, thus the same Term cannot be at the same time Hypernym and Hyponym.

It is easy to see how functional requirements are fulfilled with the generalization/specialization relationship being disjoint meaning a term to be suitable to belong to the “Glossary system” is either to be, and can only be, hypernym or hyponym.

It is also important to show how all Terms related to the Glossary-which is a Singleton class (only one instance of Glossary class in the system) need to be either hypernyms or hyponyms. Perhaps, as all words written in SRP definition are not part of the Glossary, in a more global Specification context, Term class would be a generalization of, for instance, Word class. However, addressing this issue, at this point the specification of the Conceptual Model must be related to the functionalities and goals in which this project is spanned.

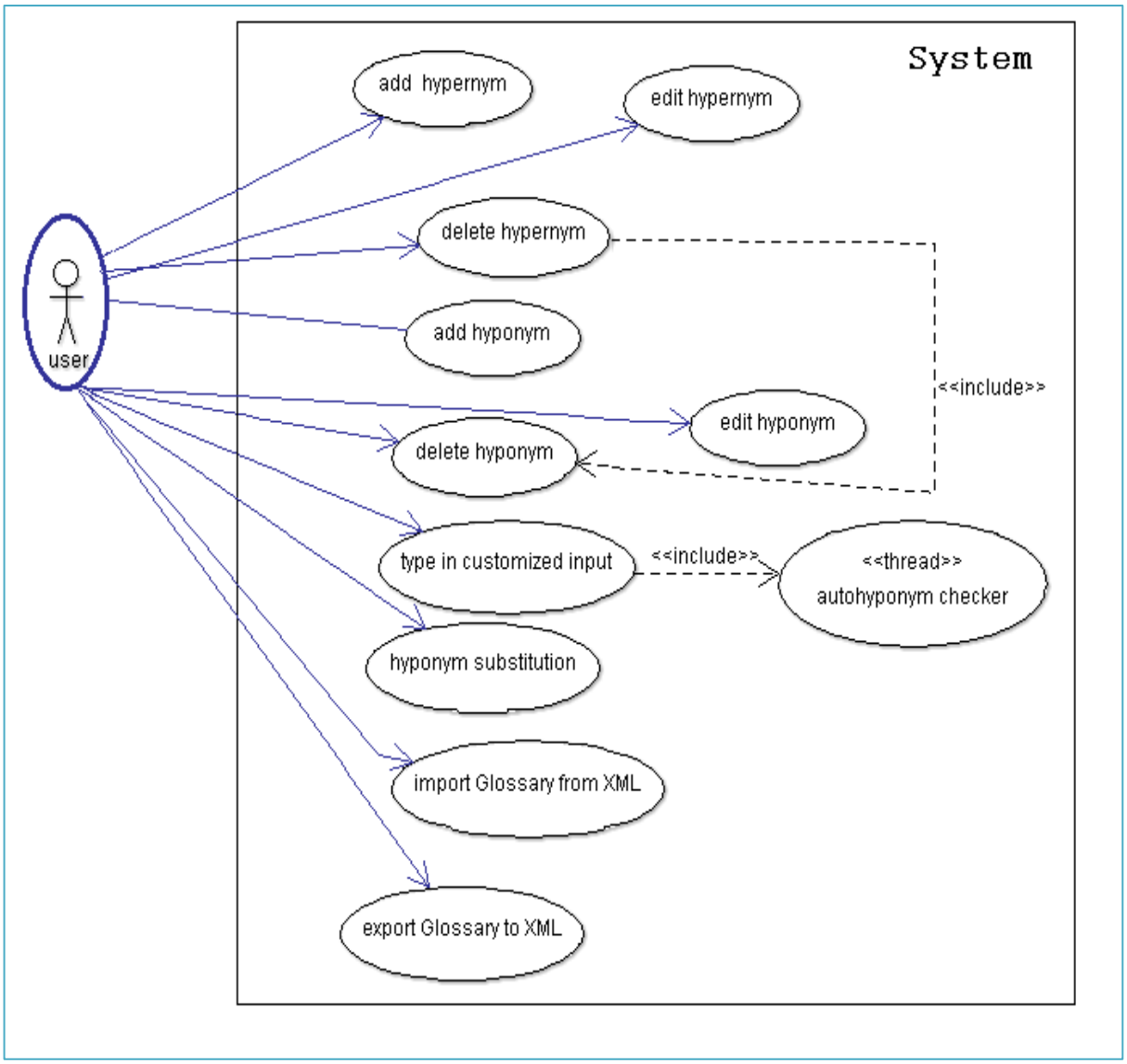
As stated in the introductory section, hypernyms must have a description, although null or void values are accepted for it. Hypernyms may also not be related to a hyponym, as clearly shown in the multiplicity of the hyponym role part on “is related to” relationship.

In the Term class, the name attribute - of class String- indicates the content of the term, that is, the value of the written or edited word.

5.2 Use Case Model

A use case is a list of steps, defining relations between an actor and a system, in order to achieve a goal, or milestone. As mentioned previously, the system's main User is the Requirements Engineering Expert which triggers almost every action leading to a goal: deciding which terms are hyponyms, which are hypernyms, associating a textual description to hypernym, writing as normal in a customized input which mocks the major part of behaving of a normal input textbox but triggers an auto hyponym checker while the user writes in it (similarly to the spelling checking of a word processor), decide to substitute –or not- hyponyms, and selecting a file to load or save the working Glossary.

With all, the following model is driven:



Dealing with the above use cases there can be a clear segmentation into three different types of use cases. For instance the addition, edition and deletion of hypernyms and hyponyms could be tagged as Glossary management; the type of customized input ,Hyponym Substitution, and auto hyponym checker could be segmented as SRP text editor management; finally the import/export the Glossary could be segmented as External Data Services.

Describing the use cases textually in more depth and according to standards defined in [Lar03] and [DSS]:

<table>
<tr><th colspan="2">USE CASE : Add Hypernym</th></tr>
<tr><td colspan="2">ACTORS User (starter)</td></tr>
<tr><td colspan="2">PURPOSE: Insert a hypernym, its description and its list of hyponyms into the Glossary.</td></tr>
<tr><td colspan="2">SUMMARY: The User provides all the required data. The system verifies the data and performs the insertion if requirements are fulfilled.</td></tr>
<tr><td colspan="2">TYPE: Glossary management -Primary and essential</td></tr>
<tr><td colspan="2">CROSS REFERENCE -</td></tr>
<tr><td colspan="2">TYPICAL COURSE OF EVENTS</td></tr>
<tr><td>ACTORS ACTIONS</td><td>SYSTEM RESPONSE</td></tr>
<tr><td>1. Use case begins when the user has provided the data and confirmed the action</td><td></td></tr>
<tr><td></td><td>2. The system receives data, and performs a first analysis. If it finds repeated values in the hyponym list it merges it into a set of hyponyms.</td></tr>
<tr><td></td><td>3. With all valid data the system checks for the functional requirements to be met.</td></tr>
<tr><td></td><td>3b. Some data violates the requirements and an error message must be shown to inform.</td></tr>
<tr><td>4b. User accepts error message dialog and proceeds to refill data and act as in point 1.</td><td>4. Data is Ok. Insertion is made correctly. System shows Glossary with new hypernym, along with its hyponyms and description. No further message is shown.</td></tr>
<tr><td colspan="2">ALTERNATIVE COURSE: Point 1- User cancels action, thus no action continues in the Use Case</td></tr>
</table>

USE CASE : Edit Hypernym	
ACTORS User (starter)	
PURPOSE: Edit a hypernym, its name, its description or its list of hyponyms, into the Glossary.	
SUMMARY: The User indicates he wants to edit one given hypernym and provides all the required data. The system verifies the data and performs the edition if requirements are fulfilled by the new data.	
TYPE: Glossary management - Primary and essential	
CROSS REFERENCE -	
TYPICAL COURSE OF EVENTS	
ACTORS ACTIONS	**SYSTEM RESPONSE**
1. Use case begins when the user has activated the edition event pinned to a certain hypernym , provided the new data and confirmed the action	
	2. The system receives data, and performs a first analysis. If it finds repeated values in the hyponym list it merges it into a set of hyponyms.
	3. With all valid data the system checks for the functional requirements to be met.
	3b. Some data violates the requirements and an error message must be shown to inform.
4b. User accepts error message dialog and proceeds to refill data and act as in point **1**.	**4.** Data is Ok. Edition is made correctly. System shows Glossary with edited hypernym information. No further message is shown.
ALTERNATIVE COURSE: Point 1- User cancels action, thus no action continues in the Use Case	

USE CASE : Delete Hypernym	
ACTORS User (starter)	
PURPOSE: Delete a hypernym and all information associated to it from the Glossary.	
SUMMARY: The User indicates he wants to delete a given hypernym.	
TYPE: **Glossary management** - Primary and essential	
CROSS REFERENCE This Use Case includes Delete hyponym Use Case	
TYPICAL COURSE OF EVENTS	
ACTORS ACTIONS	**SYSTEM RESPONSE**
1. Use case begins when the user has activated the deletion event pinned to a certain hypernym.	
	2. The system deletes all data from implicit hypernym. No message is shown.
ALTERNATIVE COURSE: -	

USE CASE : Add Hyponym	
ACTORS User (starter)	
PURPOSE: Add a hyponym to a certain hypernym, into the Glossary.	
SUMMARY: The User indicates he wants to add a hyponym of a given hypernym and provides the hyponym's name. The system verifies the data and performs the insertion if requirements are fulfilled.	
TYPE: **Glossary management** - Primary and essential	
CROSS REFERENCE -	
TYPICAL COURSE OF EVENTS	
ACTORS ACTIONS	**SYSTEM RESPONSE**
1. Use case begins when the user has selected a certain hypernym and activated the event which enables to add a hyponym to that hypernym.	
	2. The system analyzes the name given to the hyponym. If it fulfills requirements it adds the hyponym to the implicit hypernym and no further message is shown.
	2b. The Hyponym's name violates some constraint within the requirements. An error message is shown indicating it.
3b. User accepts error message and returns to Point **1.**	
ALTERNATIVE COURSE: Point 1- User cancels event and the Use Case ends.	

USE CASE : Edit Hyponym	
ACTORS User (starter)	
PURPOSE: Edit a hyponym's name, into the Glossary.	
SUMMARY: The User indicates he wants to edit a hyponym of a given hypernym and provides the hyponym's new name. The system verifies the data and performs the edition if requirements are fulfilled.	
TYPE: **Glossary management** - Primary and essential	
CROSS REFERENCE -	
TYPICAL COURSE OF EVENTS	
ACTORS ACTIONS	**SYSTEM RESPONSE**
1. Use case begins when the user has selected a certain hypernym and activated the event which enables to edit a hyponym to that hypernym.	
	2. The system analyzes the new name given to the hyponym. If it fulfills requirements it modifies the hyponym's name to the new value of the implicit hypernym and no further message is shown.
	2b. The Hyponym's new name violates some constraint within the requirements. An error message is shown indicating it.
3b. User accepts error message and returns to Point **1.**	
ALTERNATIVE COURSE: Point 1- User cancels event and the Use Case ends.	

USE CASE : Delete Hyponym	
ACTORS User (starter)	
PURPOSE: Delete a hyponym to a certain hypernym, from the Glossary.	
SUMMARY: The User indicates he wants to delete a given hyponym from a certain hypernym.	
TYPE: **Glossary management** - Primary and essential	
CROSS REFERENCE This Use Case is included by Delete hypernym Use Case	
TYPICAL COURSE OF EVENTS	
ACTORS ACTIONS	**SYSTEM RESPONSE**
1. Use case begins when the user has activated the deletion event pinned to a certain hyponym from a given hypernym.	
	2. The system deletes all data from the hyponym of the implicit hypernym. No message is shown.
ALTERNATIVE COURSE: -	

USE CASE : Type in Customized Input	
ACTORS User (starter)	
PURPOSE: Allow to write text in a customized input which enables hyponym highlighting.	
SUMMARY: The User indicates he wants to delete given hypernym.	
TYPE: **SRP text editor management** - Primary and essential	
CROSS REFERENCE This Use Case includes Auto hyponym Checker Use Case *Note*: This Use Case extends from all insertions of Forms, Metrics, Patterns and Dialogs which formed part of other past projects and are part of the PABRE-Man system.	
TYPICAL COURSE OF EVENTS	
ACTORS ACTIONS	**SYSTEM RESPONSE**
1. Use case begins when the user has written something in the customized input text box situated in the different forms of SRP definitions.	
	2. The system actually shows each letter written by the user in the text box area the same way a normal text box would do. If the user writes an invalid word it highlights it. No further system message is shown.
ALTERNATIVE COURSE: -	

USE CASE : Auto hyponym Checker	
ACTORS System <<thread>> stereotype	
PURPOSE: Analyzes the text in a customized input to highlight invalid words	
SUMMARY: A Batch light-process analyzes the customizes input text every time the user writes in it to search for invalid words which to highlight without any action being done to this purpose.	
TYPE: **SRP text editor management** - Primary and essential	
CROSS REFERENCE This Use Case is included by Type in Customized Input Use Case	
TYPICAL COURSE OF EVENTS	
ACTORS ACTIONS	**SYSTEM RESPONSE**
	1. Each time a word or letter is written or content is pasted in the custom text editor this process is fired. It analyzes all the text, and dies.
ALTERNATIVE COURSE: -	

USE CASE : Hyponym Substitution	
ACTORS User	
PURPOSE: Substitution of hyponym by its hypernym	
SUMMARY: User is informed of an invalid hyponym word. He selects to substitute it with the word's hypernym and the system does so.	
TYPE: **SRP text editor management** - Primary and essential	
CROSS REFERENCE -	
TYPICAL COURSE OF EVENTS	
ACTORS ACTIONS	**SYSTEM RESPONSE**
1. **User is fully informed on all the hyponyms in the editable text. User selects one of them and initiates the action to substitute the single instance of it or all the instances of that hyponym in the text.**	
	2. System receives hyponym , text and replacement mode and substitutes the hyponym by its hypernym in the text the number of times replacement mode indicates
3. **User continues either writing or performs again point 1**	
ALTERNATIVE COURSE: -	

USE CASE : Import Glossary from XML	
ACTORS User	
PURPOSE: Provide an importation mechanism to build a Glossary from a XML file.	
SUMMARY: User chooses a file, system checks file is well formed, parses and forms the implicit Object from the file's contents.	
TYPE: **External Data Services** - Primary and essential	
CROSS REFERENCE -	
TYPICAL COURSE OF EVENTS	
ACTORS ACTIONS	**SYSTEM RESPONSE**
1. **User clicks on menu according to load a file, chooses file from file system and presses OK.**	
	2. System reads file from the given path and parses it according to a parser XML modeler.
	2b. System detects error in reading file, or the XML is badly formed according to the XML modeler. An error dialog showing this error is shown.
3b- User accepts error dialog, tries to resolve issue, and returns to Point **1.**	
	3. On success, the system loads the imported Glossary as the active one in the system. A success message is shown indicating the active Glossary has been loaded from the file.
ALTERNATIVE COURSE: - Point 1 – User cancels file selector , thus Use Case ends	

USE CASE : Export Glossary to XML	
ACTORS User	
PURPOSE: Provide an exportation mechanism to save a Glossary to a XML file.	
SUMMARY: User chooses a filename and path and saves the working Glossary to a legible XML file according to standards defined by XML modeler.	
TYPE: **External Data Services** - Primary and essential	
CROSS REFERENCE -	
TYPICAL COURSE OF EVENTS	
ACTORS ACTIONS	**SYSTEM RESPONSE**
1. **User clicks on menu to save Glossary, chooses file location from file system, gives a name to the file and presses OK.**	
	2. System saves the Glossary to the given file in the given path parsing it according to a parser XML modeler.
	2b. System detects error during the parsing step or other general error- for instance non permission to save file – and an error message is shown accordingly.
3b- User accepts error dialog, tries to resolve issue, and returns to Point **1.**	
	3. On success, the system saves the Glossary on the given file, and on the given path.
ALTERNATIVE COURSE: - Point 1 – User cancels file selector , thus Use Case ends	

With the implementation of the different Use Cases the following sub-objectives are covered up:

- **Adding/Editing/Deleting glossary entries**
- **Hyponym detection**
- **Morphism Application**

5.3 XSD and example

As mentioned in above in the External Data Services Use Cases, the system uses an XML parser modeler to validate XML files which store the representation of the information stored in a Glossary and also uses the modeler to save the Glossary in XML format. The way to implement this action is using XML Schema. XML Schema sometimes called WXS, but generally referred to it as XSD, is also an XML which consists of a set of rules an XML document must conform in order to be valid. XSD has an own and proper language with special tags and nameset with restricted words indicating schema components , types or assertions made to sets found in XML declaration. For example in XSD there can be Element declarations, Attribute declarations, Simple and complex type declarations and restrictions ,lists or unions on sets. [XML]

For the Glossary Object the following XSD is used:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
	<xsd:annotation>
		<xsd:documentation xml:lang="en">
			Schema that determines the structure of XML documents
			with information about the glossary of terms.
		</xsd:documentation>
	</xsd:annotation>

	<xsd:element name="Glossary">
		<xsd:complexType name="entry">
			<xsd:sequence>
				<xsd:element name="hypernym" type="xsd:string"/>
				<xsd:element name="description" type="xsd:string"/>

				<xsd:complexType name ="hyponyms">
		<xsd:element name="hyponym"  minOccurs="0" maxOccurs="unbounded"
type="xsd:string"/>
				</xsd:complexType>

			</xsd:sequence>
		</xsd:complexType>
	</xsd:element>
</xsd:schema>
```

The Schema begins with an annotation with an explanation on the schema. It then defines that the first element of a valid XML-Glossary file must begin with the tag “<Glossary> ... </Glossary>” in that exact manner. The same happens with the entries of the glossary which are a sequence of type-structures containing two strings – hypernym and description- and an unbounded list of complex-type which represents the set of hyponyms to the given hypernym - in this case a list of string elements- .

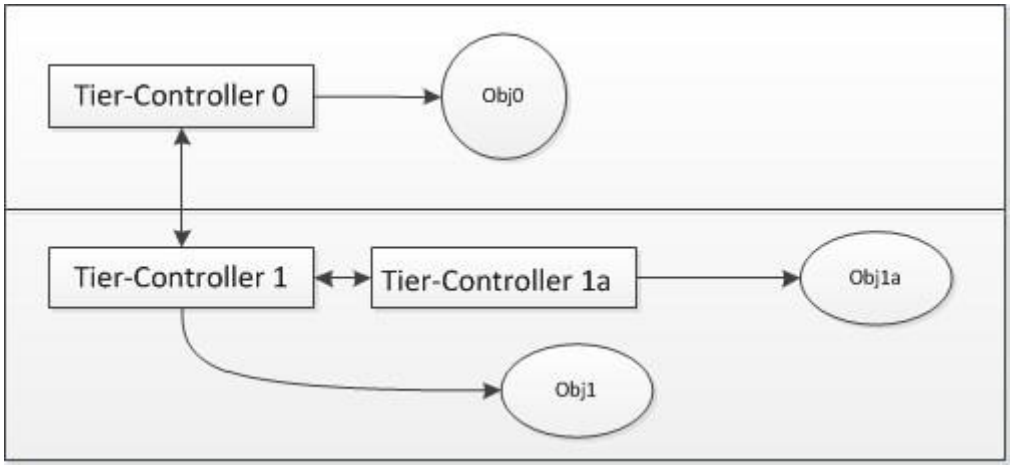
A simple example which fulfills the above Schema is the following content of a possible XML file where a glossary is stored:

```
<Glossary>
 <entry>
  <hypernym>OPERATING SYSTEM</hypernym>
        <description>A DESCRIPTION</description>
              <hyponyms>
                    <hyponym>OS</hyponym>

              </hyponyms>

  </entry>

       <entry>
  <hypernym>PASTA</hypernym>
        <description>DELICIOUS! </description>
              <hyponyms>
                    <hyponym>SPAGHETTI</hyponym><hyponym>RAVIOLI</hyponym>
                    <hyponym>FARFALLE</hyponym><hyponym>MACARRONI</hyponym>
                    <hyponym>FETTUCCINE</hyponym>

              </hyponyms>
  </entry>

       <entry>
  <hypernym>frutas</hypernym>
        <description>Deliciöso! </description>
              <hyponyms>

       <hyponym>pera</hyponym><hyponym>plátano</hyponym><hyponym>limón</hyponym>
                    <hyponym>naranja</hyponym><hyponym>fresas</hyponym>

              </hyponyms>
  </entry>

  <entry>
  <hypernym>flowers</hypernym>
  <description>smell good</description>
  <hyponyms>
   <hyponym>rose</hyponym>
   <hyponym>petunia</hyponym>
   <hyponym>lilly</hyponym>
  </hyponyms>
 </entry>
</Glossary>
```

Notice that all-capitalized letters and terms with midterm spaces are allowed as well as letters with Spanish, Catalan, and French symbols. The hyponym list may also be included in one or various lines; there are no restrictions about this except that its tags ought to be well formed.

6 Design

Design section on this project turns out to be critical. An inefficient design can penalize productivity in such a manner to slow down the whole system. A high, or tightly, coupled design can report a too excessive payload on reusability and can cause a “ripple effect” on a module change, plus increase inefficiency as a side effect. Additionally to this considerations there must be a design constrain upon the maintenance of the Three-Tier Architecture at all levels. Three-tier architecture is typically composed of a presentation tier, a business tier, and a data tier. Further on, functionalities must be monitored by a controller which interacts with the various objects it controls and along with different controllers. Graphically:



Notice that if Obj1a wants to access a method or public attribute from Obj1 it has to go through its own controller – Tier-Controller 1a – and then Tier-Controller 1 to access Obj1 and its desired artifact.

PABRE system also is based upon Domain Model pattern as a design to the business tier. Objects or classes perform their respective operations separating functionalities and responsibilities amongst objects each other and between them. This is also known as the low coupling and high cohesion principle.

Joining both Three-Tier Layer and Domain Model architecture the outcome turns out to be a system which puts the responsibility of functionality and calculations in the different Domain Objects, and the results of these calculations together with the communication among objects and tiers is delegated to the controllers. Less imbricated controllers tend to be tier controllers, which follow the Facade pattern being the only point of entrance of the flow in the tier and providing all functionality to the accessed tier.

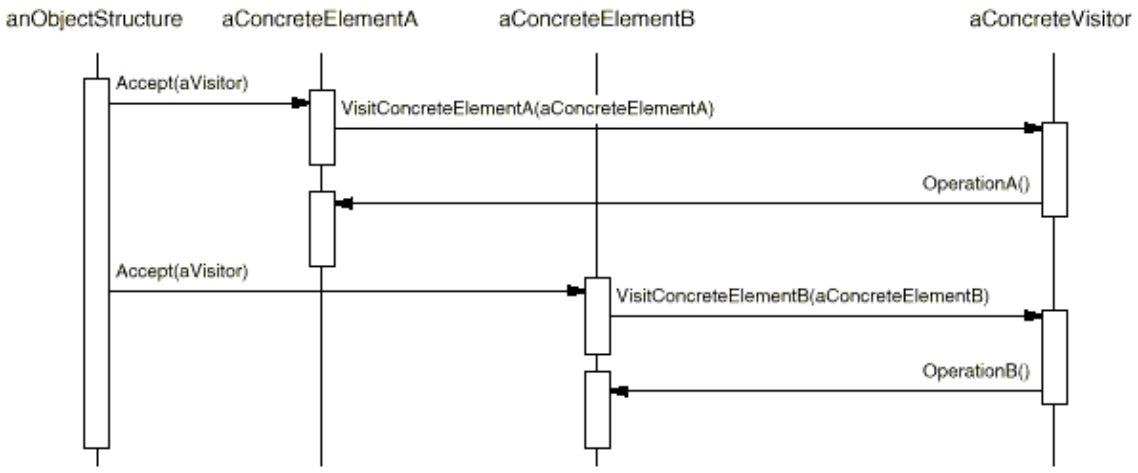
6.1 Guideline to Solution

Without considering the above constraints, the first thought to reach the global goals is thinking about the presentation tier. It is easy to see that the presentation tier is, actually, very similar to word processors' syntax and grammar corrector, underlining invalid words and offering a new window interface suggesting corrections upon a list of high rated candidate words respect to a certain invalid word. Relying upon the interface, and applying the criteria based on the non-functional requirement that states that the interface must be quick and usable, this new window to analyze invalid words must be simplified to a fast an quick User Interface, perhaps a pop-up under the right click event. Red underlining seems a valid way to highlight an invalid word.

As a reference guide to arrive to a solution to a similar situation "Design Patterns , Elements of Reusable Object-Oriented Software" from Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides [GoF] in its section 2.8.

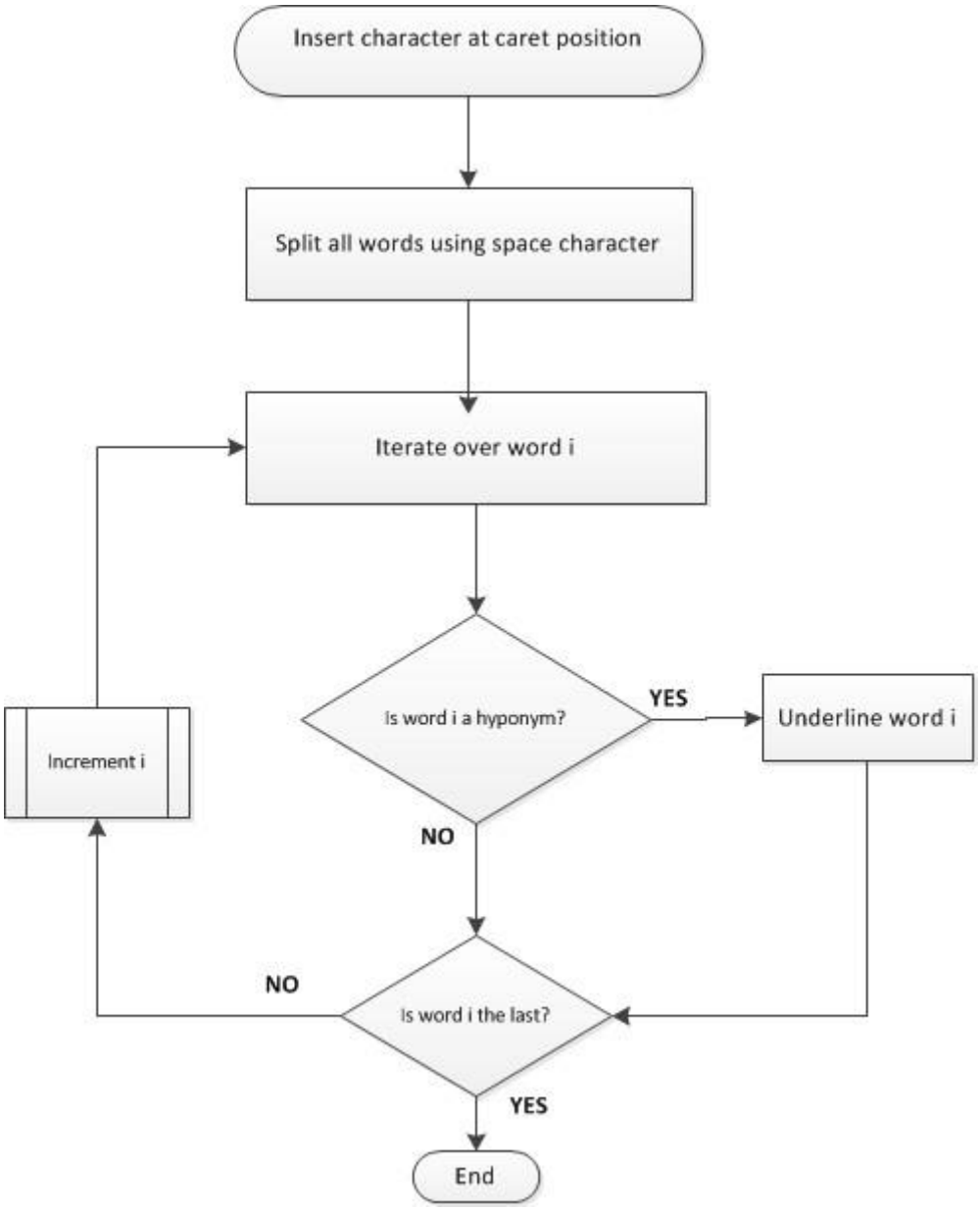
In order to access to the data in the customized Glyph case, the Iterator pattern is used and the Abstract Factory pattern is used to render the different look-and-feel from the different User Interfaces. Skipping these aspects due to the fact that they are already fixed by the natural UI look-and-feel associated to PABRE system and the organization of the documents is implicit to Swing Java library.

Further on the syntactic analysis is encapsulated within the Visitor pattern. This pattern acts over the String Object that represents the written input channel whenever the user types a letter. Visitor pattern follows the structure below:



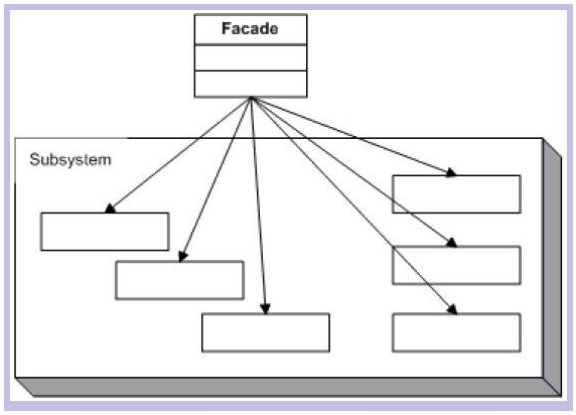
In the case of this project, Concrete Elements are String Objects – either one char as typed by the user, or a chunk of copied (and unanalyzed) text or a word processed by an intermediate split with the purpose of whole analysis- which have an acceptance operation of being a hyponym or not.

With respect to the application of the Visitor pattern to this project it must follow the following flow process:

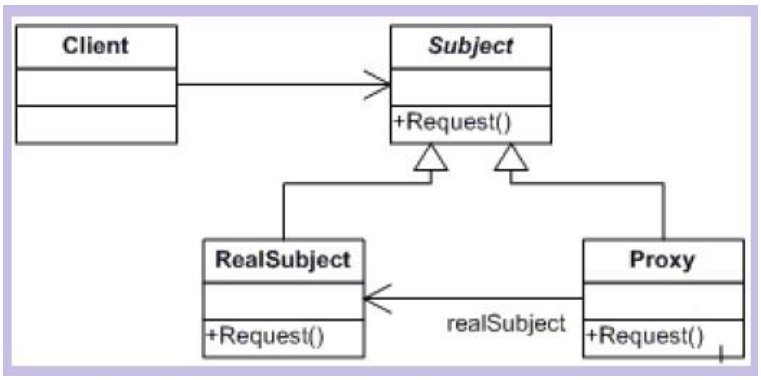


As mentioned before, on the first state, a character may also include a blob of text copied in the clipboard. It is obvious that checking if a word is or not a hyponym has to undergo a call to the business tier, properly bypassed by the correct Controllers, as the query has to be made to the actual and implicit dictionary.

The domain tier, previously defined in the Specification chapter of this document under the Conceptual Model section, must be designed further on following the requested architecture. Moreover, being the Glossary Domain Controller the entrance point for each function of the Glossary, it meets all the conditions to fulfill the Facade and Proxy Patterns.

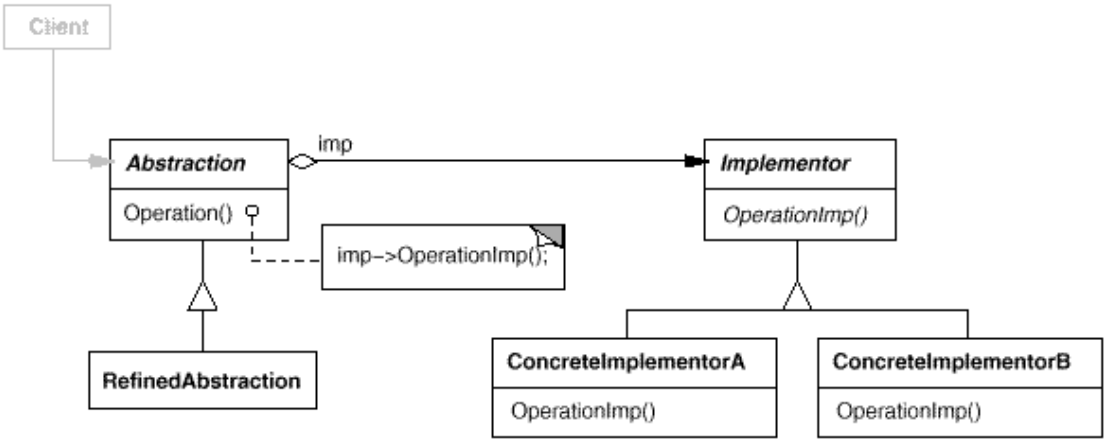


Facade Pattern



Proxy Pattern

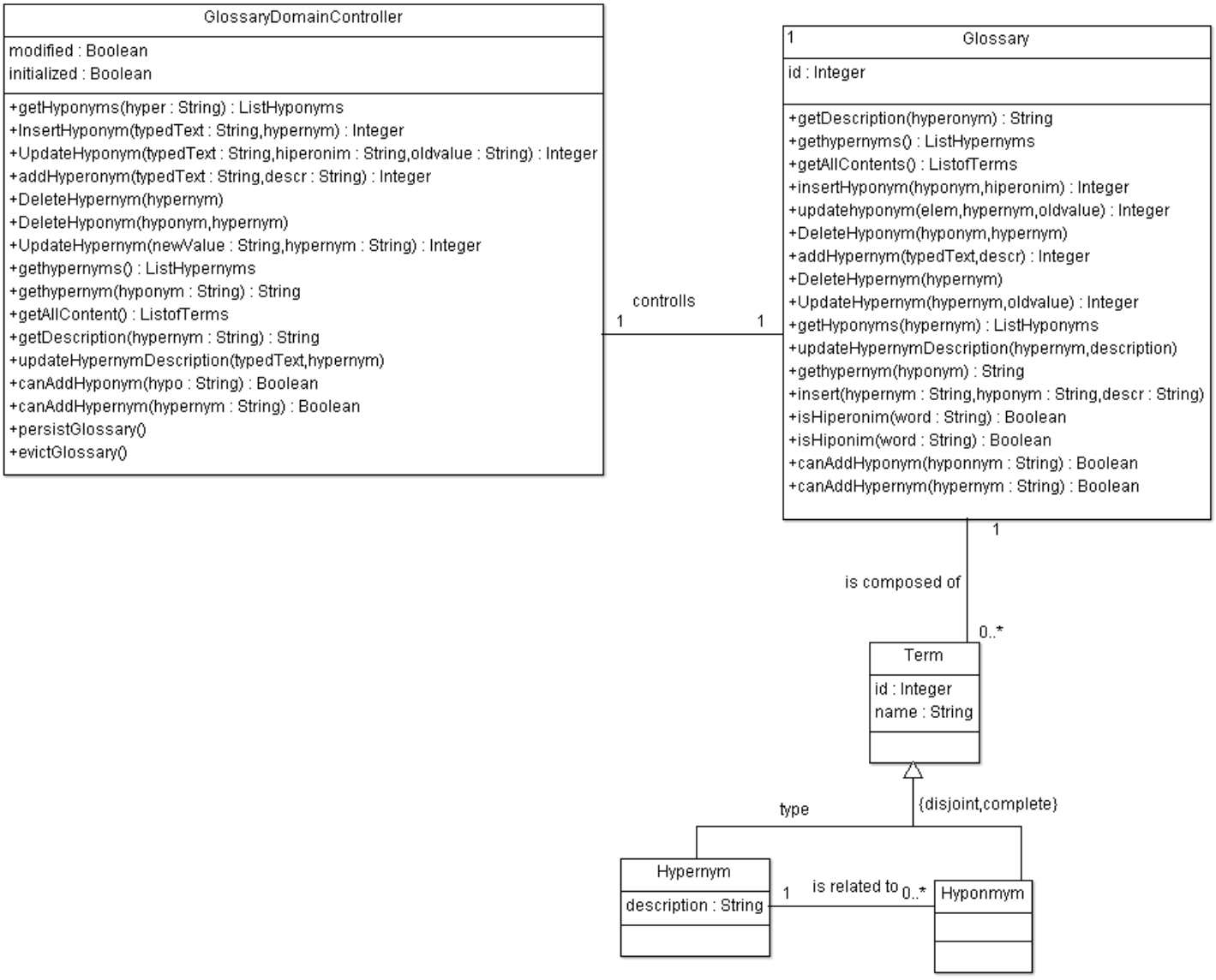
In the same direction, and within the Three Tier Architecture pattern, the Product Owner has stated its will to persist the Glossary using both the file importation/exportation system and the database management system. This situation suggests applying another Software Pattern to resolve and adapt to this scenario. The Bridge Pattern accomplishes this purpose as is easily seen in the following diagram:



Concrete implementations in this project include operations that implement persistence in XML file format as a method of importation and exportation of the data and other operations that implement persistence using a Database Management System.

6.2 Design Conceptual Model

As a result of the application of above patterns, altogether with the requirements for this project, and refining the Specification Conceptual Model designing its operations, the following Conceptual Model is obtained:



6.3 Design alternatives

Even though all the different requirements, architecture and pattern design do pin off degrees of freedom in order to design the resultant system, there are certain issues where a decision needs to be made. This is the case of the way data will be persisted into our system. It is crystal clear how PABRE-Man persists its data using Hibernate to access the Database Management System – in the time of the realization of this project it was Apache Derby- but, since the Glossary is semantically independent from the SRP Catalogue and other sub-system in PABRE-Man, should it stick with Hibernate persistence? Could the Glossary be saved by other storage system?

It is noticeable to remark the difference between persistence and importation / exportation. When dealing with persistence it is assumed the persisted data is vital for the flow of the application. It could be seen as that the data needed to run for the first time the application will need to be persisted somehow. Importation or exportation could be seen as a snapshot of the data of the implicit object, which in turn could have been filled up by persisted data.

The decision of how to persist the Glossary covers up three different options: Java Serialized, XML file, and Hibernate.

6.3.1 Java Serialized

Java provides a method, called object serialization, where an object is represented as a sequence of bytes. Object's data as well as its type and the types of the object's attributes are stored sequentially (marshalled) in a given file. After a serialized object has been written into a file it can be read as a normal file, and with the file's stream the object can be obtained again , after a process of un-marshaling , that is, using the type information and bytes that represent the object and its data which can be used to recreate the object in memory. Most surprising is that the process, as a whole, is independent, meaning that an object can be serialized on one platform and deserialized on a different platform.

Files which contain Serialized objects are illegible when opened with a word processor either to view or to be written to, and that is a very important disadvantage. In fact, if a file with a serialized object is written to any number of characters when opened by a word processor, the file is corrupted when trying to read it and un-marshal the object from it. For the Glossary to save and load data it is a must to be able for the User to identify entries and be modify or create new ones from an external Application in the persisted artifact.

6.3.2 XML-File

Similarly to the method mentioned in XSD Section, using the SAX Library a Schema File can be defined so it can be followed to build a persistence XML- file with tags which can be recognized and replicated by the User in order to add, edit or delete entries with another Application, perhaps a text processor. This persistence file may also be read as a normal text file and be precisely parsed into a Glossary object just following again the rules defined in the Schema File. If errors are found, for instance, because of non-expected characters found or the persistence file was not loaded following the given Schema, error handling identifies the error type and a customized error messages can be shown. This really gives the User a good feedback on how and what to fix in order to get the system working. [10]

6.3.3 Hibernate

As mentioned in Technology section, Hibernate is a library which helps save Objects into Relational Databases. It is the option that is used broadly in the PABRE system in order to save Business Tier Objects that needs to be persisted. User or developer can access data through a SQL client (such as SQuirreL or Toad) to see data or perform update or insert SQL queries to modify it. This seems a bit tedious, especially for the User, because the process of updating and inserting large quantities of data is pinned up to knowledge of SQL instructions and procedure.

Before all, hibernate must be tuned up correctly in order to align with PABRE-Man Hibernate configuration and mapping. The configuration is basically the same, due to the fact that in the integration process it would be wise to use the same Database Management System that is configured already. Mapping, however, has to be modified by adding the following code to Mapeig.hbm.xml file (which is called by the configuration XML file). [HIB 1,2,3]

```
<class name="edu.upc.gessi.rptool.domain.glossary.Glossary" lazy="false" table="GLOSSARY"  >
<id name="id" column="id" type="integer" >
<generator class="native"/>
 </id>
        <map name="glosary" table="SYNONYM">
                 <key column="IDGLOSSARY" not-null="true"/>
                <index column="HYPONYM" type="text"/>
                <element column="HYPERNYM" type="text" not-null="true"/>
        </map>
         <map name="Description" table="HYPERNYM_DESCRIPTION">
                 <key column="IDDESCRIPTION" not-null="true"/>
                <index column="HYPERNYM" type="text"  />
                <element column="DESCRIPTION" type="text" not-null="true" />
        </map>

</class>
```

Notice a whole Glossary object is saved. The Glossary object contains a map named glossary which is mapped in the SYNONYM table with key IDGLOSSARY- indicating the id of the Glossary Object in which it is mapped- and an index column named HYPONYM and an element column named HYPERNYM. This follows the idea that a pair <hyponym, hypernym> cannot be repeated. Similarly happens with the Description object mapped in the HYPERNYM_DESCRIPTION table, where DESCRIPTION is stored in a same record with a HYPERNYM which cannot be repeated. This warrants the accomplishment –in a data layer perspective- of the functional requirements initially defined.

6.3.4 Selected Option

A decision must be taken in order to design a proper and efficient persistence mechanism. For that reason further testing has been made to analyze how each option behaves in temporal and disk consumption axis. Taking as a reference the code which corresponds to the above Hibernate mapping, having a <String, String > Hashmap representing the glossary and a <String, [String] + > Multimap representing a <hypernym, <Hyponym List> > type structure named hyponyms (used to increase granularity for controller data transfer and thus increment the final efficiency of the system) the results obtained are the following:

/w/	20	2000	4000	5000	10000	50000	100000
Time							
Hash Map	87	966	2308	3095	5311	20560	41282
Multi-map	9	689	954	1102	2197	10955	21331
XML	145	1768	2015	2304	2491	5647	10387
Total Serialized	96	1655	3262	4197	7508	31515	62613
File size							
Serialized	1	73	147	184	370	1932	3883
XML	3	362	725	907	1815	9237	18514

W is the cardinality of the glossary, time is measured in milliseconds and File Size in KB. The times of the Hashmap and Multi-map correspond to the times taken by a serialized process of saving and loading the respective object.

For instance, for a glossary of 5000 Terms, the time to save the Hashmap associated to the glossary was approximately 3 seconds, and the time to save the Multi-map was a bit more than a second, so the total time to save the object is about 4.2 seconds. The time spent to save the contents of the object into an XML file is 2.3 seconds. The disk space cost of the operation results in a Serialized 184 KB file and an XML file of about 1 Mb.

The test is repeated under the same conditions to test the Hibernate option. Same size as before glossaries are sent to save from a flushed table and loaded from previously saved tables (only measuring loading times). Since the essential information is held in the Hashmap, this is the only object to be saved or loaded. Hence, an additional timing measurement has to be made to create the hyponyms Multimap. This time the whole object is set to be saved, including the Hashmap and Multimap objects. The results are shown as follow:

/w/	20	2000	4000	5000	10000	50000	100000
Save	345	2472	4872	4623	8300	36952	81810
Load	69	16	16	16	18	18	19
Create hyponyms	4	6	12	16	17	30	56

Again, time is in milliseconds so a 5000 term glossary is saved in 4.623 seconds and loaded in 16 milliseconds, the same time hyponyms Multimap is created, giving a total object loading throughput time of 32 milliseconds.

6.3.4.1 Conclusion

Serialized option is a good option if the only purpose of it where to save the serialized object to a file and then recover it. Its saving and loading time seem reasonable but its great advantage is the optimized disk space size it uses to save files. However its null functionality to modify or view the data since it is somehow codified in serialized object byte stream file makes this option void to fulfill any goal.

XML option is great in order to visualize, edit, add, and delete the data in the glossary since it is saved in a normal stream text file with any desired encoding. Due to the fact it has to build the stream, open the file descriptor channel, and write the effective stream (possibly a huge String object) to the file, the file size variable is too big compared to the serialized option as it grows almost exponentially, and the time variable is not suitable for probable glossary sizes, although for bigger sizes it tends to stabilize.

Hibernate option is similar in timing as Serialized option but when data is saved it is possible to be viewed without decoding the data, since it is stored in a relational database and can be queried by SQL statements. However the granularity of modifications is limited to be low since a SQL process or script must be needed to insert huge quantity of data at a time.

Perhaps the solution lies in a combination of a reasonable quick load and save method and a good visualization and modification of stored data. Therefore the solution is:

- Hibernate to load and save the whole object
- XML to export and import –and then actualize saving into the database through Hibernate- data to/from a XML file.

Dealing with the initial load of the glossary in the application, using the Hibernate option the timing can be masked by the load of other application data also loaded trough Hibernate.
This resolves the following sub-objective: **Glossary persistence.**

6.4 Special design considerations

There are many business tier designs to implement a set of strings that represents a morphism application with the guidance of a dictionary. For instance a forest of two-leaved trees representing hypernyms as the root and its hyponyms on the leaves could be a solution. Implicitly Java and other Object Oriented languages tend to translate that solution to a Hashmap of references to objects, for instance called Nodes. The persistence of this solution becomes bloody difficult when persistence is implemented. Instead, and since the elementary Object of this project is the String, the Hashmap and Multimap solution seems proper and accordingly clean and easy to implement and to persist. In fact the Hibernate mapping presented above works just plain easy and only three tables need to be created in SQL creation scripts: GLOSSARY, with only an integer column, HYPERNYM_DESCRIPTION and SYNONYM tables with an integer and two varchar variables.

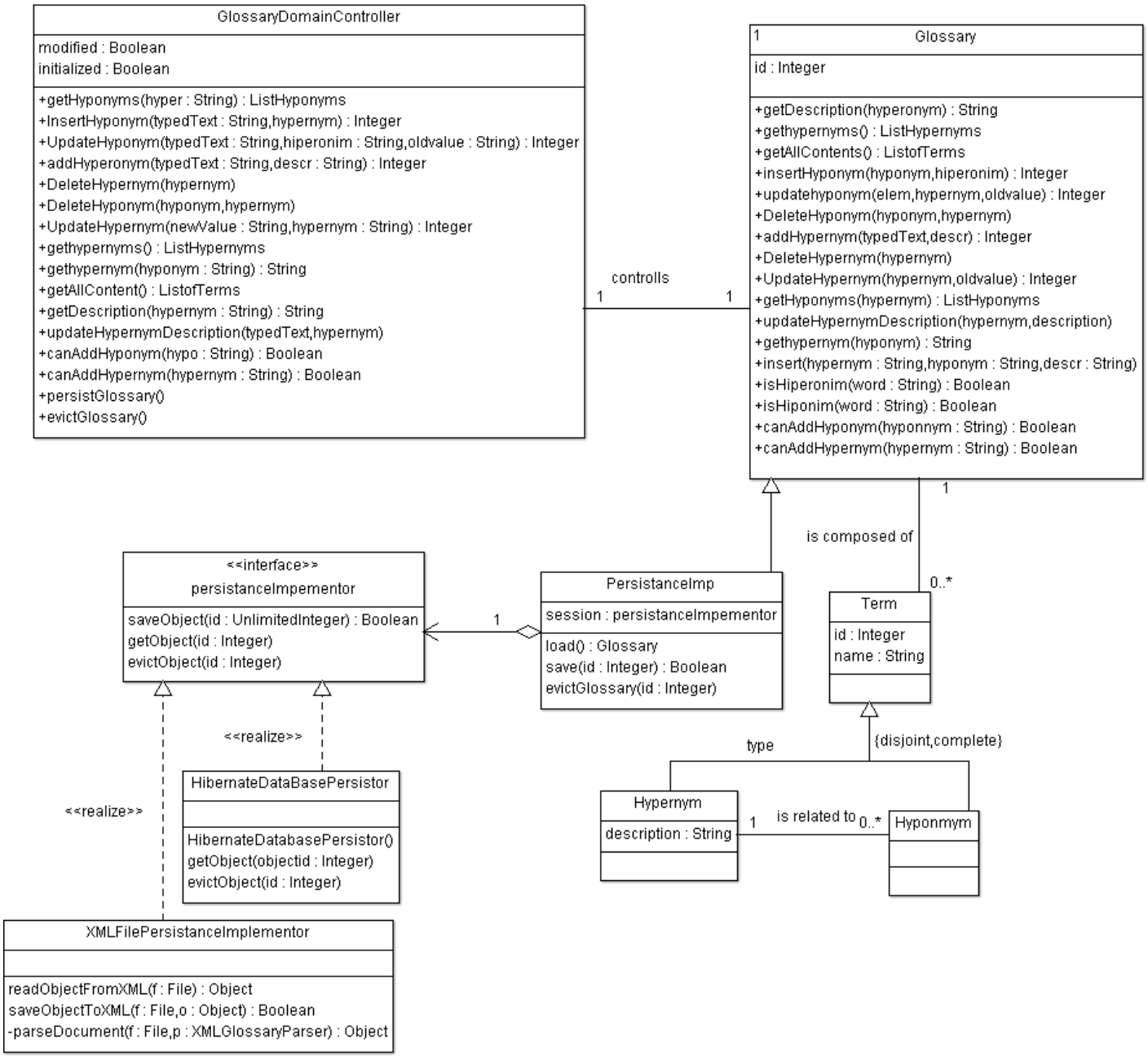
Formally in a tiny SQL table creation script:

```
create table GLOSSARY (id integer);
create table SYNONYM (IDGLOSSARY integer, HYPONYM varchar(50),HYPERNYM varchar(50));
create table HYPERNYM_DESCRIPTION
    (IDDESCRIPTION INT NOT NULL, HYPERNYM varchar(50) NOT NULL,
          DESCRIPTION varchar(1000)not null, PRIMARY KEY (IDDESCRIPTION, HYPERNYM));
```

Besides, efficiency has been increased by the usage of Hashmap and Multimap since its insertion and deletion and modification methods are very quick , in fact constant in the major part of the functionalities and at most lineal to the size of the hyponym list(not the whole glossary though). In the other hand, the Node alternative would be less efficient do to the fact it has to create every Node object, assign memory for it, and store it somehow, plus the different update and deletion methods are cost full.

6.5 Definitive Design Conceptual Model and Data Persistence Adapter

The following is the result of the design conceptual model plus the chosen persistence adapter.



6.6 Process design

Process design is important to give a final specification to functionalities of the system. Use cases are refined and linked to design artifacts to produce sequence diagrams which describe in a better way the program flow. For simplicity reasons and due to the similarity of some use cases the ones which represent the main functionalities of this project are chosen.

The following use cases are found to be significant enough to be used to define proper sequential diagrams:

- Insert Hypernym
- Hyponym Substitution
- Auto hyponym Checker

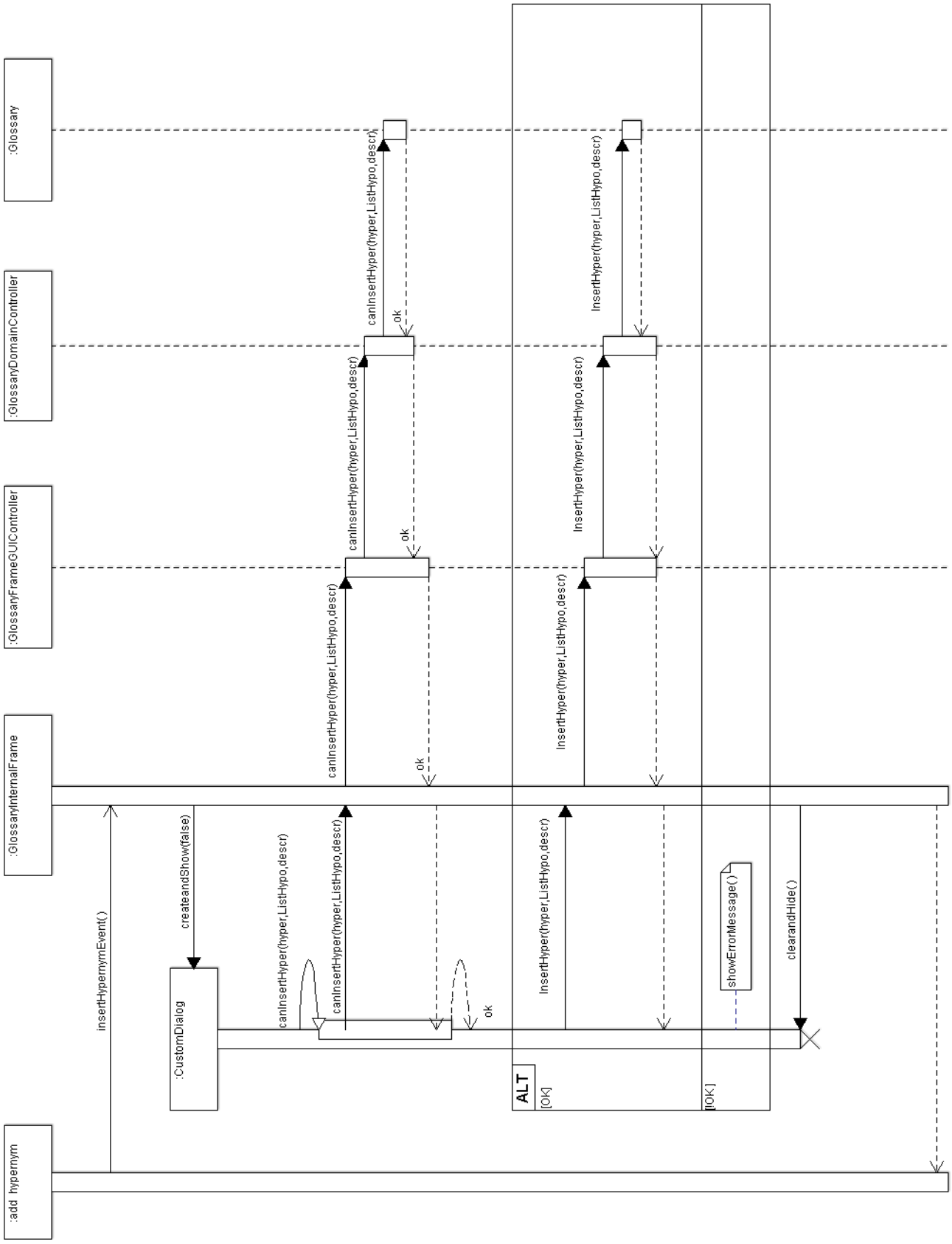
6.6.1 Sequence Diagrams

6.6.1.1 Insert Hypernym

As a brief resume, the user types in the selected hypernyms name, a list of hyponyms and a description, and if a restriction is violated an error message is shown. Otherwise both the hypernym and its list of hyponyms and description are added in the system.

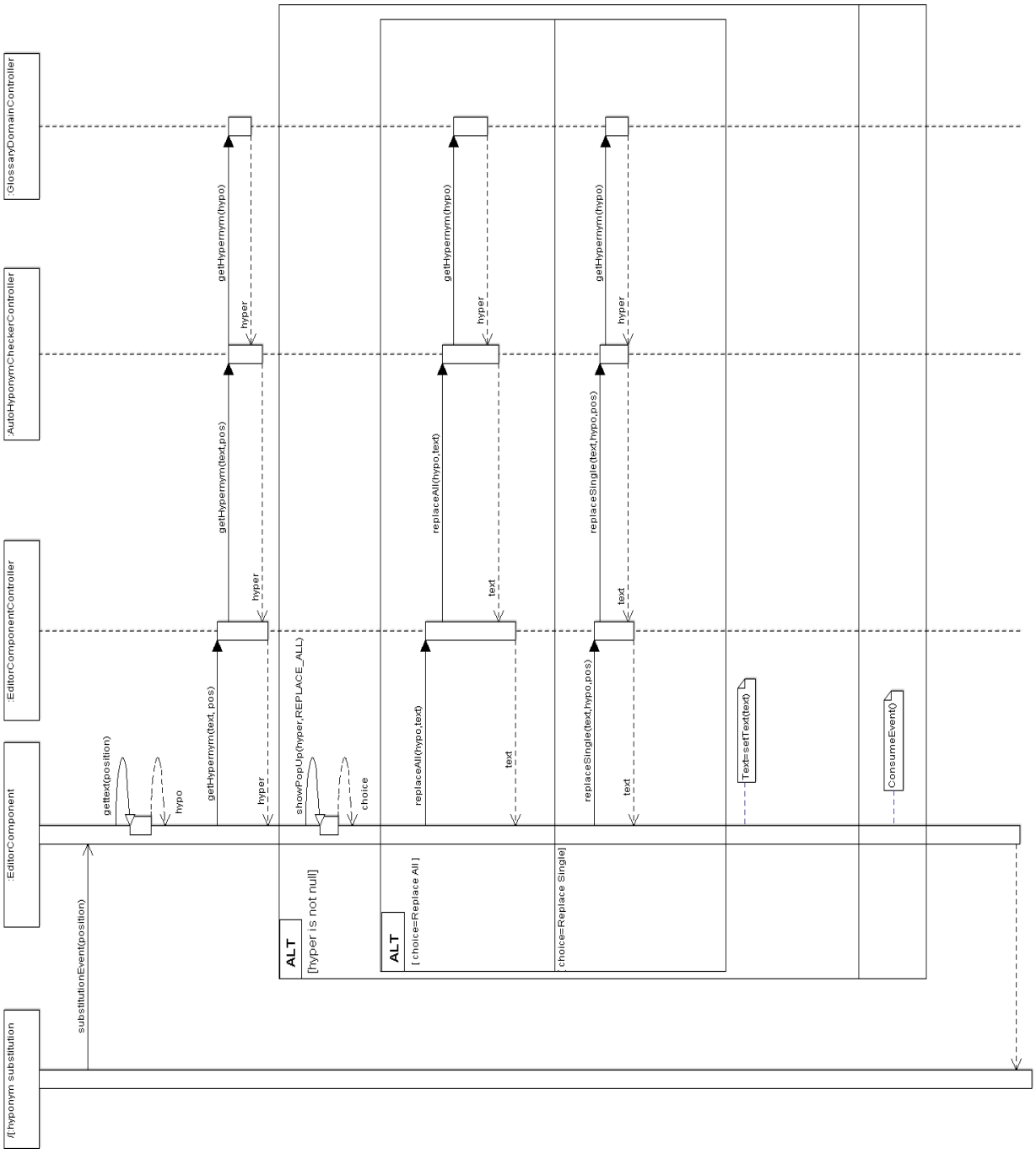
(*Note: Diagram is rotated 90º for visibility reasons*)

Notice how a dialog is created to receive the data that the user types, and destroyed after confirmation or cancellation is confirmated. When OK button of the confirmDialog is clicked the data is pushed through the controllers to check and verify for correctness. If incorrect, an error message is shown on top of the input dialog. This enables functional requirements to be fully accomplished. If not , data can be inserted directly because the checking preconditions have been satisfied.


:Glossary
:GlossaryDomainController
:GlossaryFrameGUIController
:GlossaryInternalFrame
:add hypernym
insertHypernymEvent()
createandShow(false)
:CustomDialog
canInsertHyper(hyper,ListHypo,descr)
ok
InsertHyper(hyper,ListHypo,descr)
showErrorMessage()
clearandHide()
ALT
[OK]
[OK]

6.6.1.2 Hyponym Substitution

As a brief resume, the user decides to trigger the substitution event upon the editor which contains all the text, and on a given hyponym. If the user decides to substitute the hyponym, the hyponym's hypernym must be shown as well as an option to replace all the occurrences of the hyponym in the text. Otherwise both if the user selects a non-hyponym word or decides he or she does not want to substitute the hyponym anymore, nothing has to be done.



In this sequence diagram it is shown, again, how the Three-Tier Architecture forces the communications between elements of the same Tier and definitely elements of different Tiers. It also segments responsibilities leaving a close relation between functionalities and controller responsibility. For instance AutoHyponymChecker controller is the controller which controls the functionality of the batch thread that underlines invalid words, but also controls the to remove useless punctuation marks if they are adjacent to the potential hyponym (and that would fiddle up the hyponym substitution process) and append them to the substitute hypernym as they were on the hyponym, mocking hyponym's capitalization or getting the hypernym associated to the word corresponding to the click's position or null if it does not have a hypernym. Similarly happens with the total or single replacement call, where the responsibility of finding the hypernym is left to AutoHyponymChecker controller. Despite efficiency loss – the hypernym is actually calculated as a local variable of the Editor Component class -responsibility must be respected as the functionality belongs to AutoHyponymChecker controller.

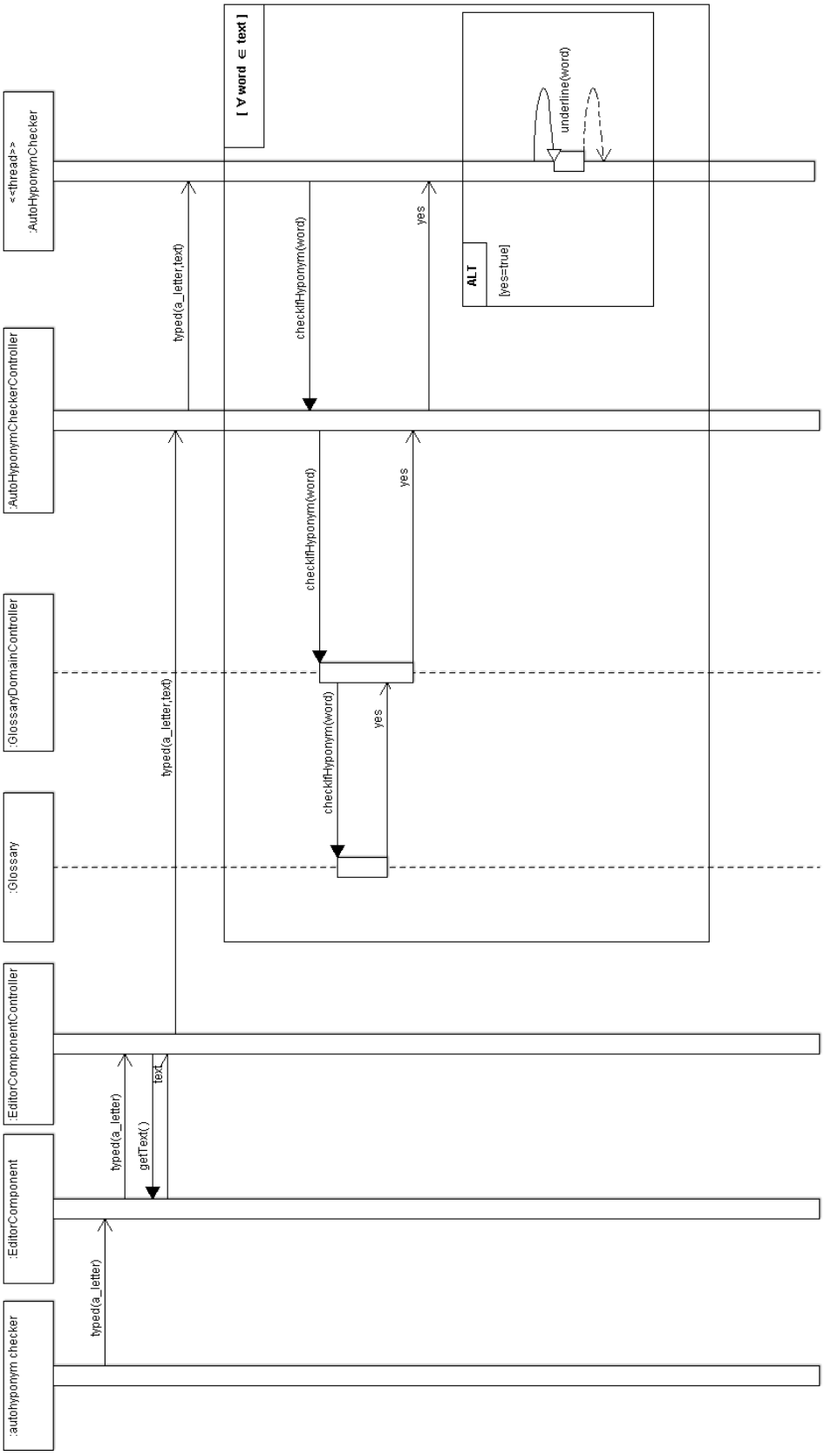
It is also remarkable how Popup menu fills the requirement of a quick intuitive user interface to perform this process but this aspect will be discussed further on in the External Design Chapter.

6.6.1.3 Auto hyponym Checker

As a brief resume, as the user writes in a customized editable text component a batch background process is executed to check if the written text contains a hyponym or not. If it does contain a hyponym it must be underlined with a red zigzag line. In this diagram it is interesting how the event is pushed through the controllers to activate the batch process. This batch process is a light thread, indeed it is a sequential iteration over a String which represents the whole text, which breaks the text into words and analyzes it word by word. Since it is a thread other actions can be made concurrently and, in the worst case, the text underlining will be updated at the next pressed key. This has to be done with the whole text as a granular entity because, for instance, pasted unchecked content can be pasted using one key press, or the text can be modified in the middle of a word or in the proper middle of the text, making the word granularity invalid.

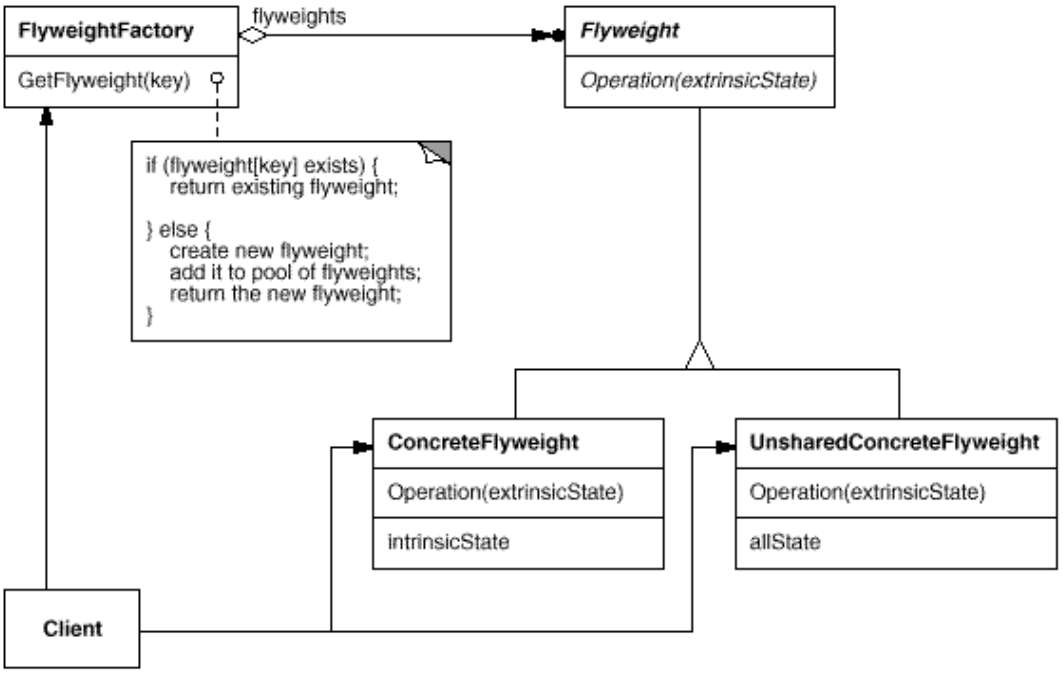
The following sequence diagram describes the whole process:

(*Note: Diagram is rotated 90º for visibility reasons*)



It is also noticeable to mention the application of the Flyweight pattern in the loop element while breaking the Editor Component's text – information retrieved via successive controllers- into words. Instead of breaking and splitting String objects into tinier String objects, Break Iterators are used as concrete Flyweight object in application of this pattern. This option makes the loop perform faster as it uses positions as references to one only text instance. Since this process has to be executed many times, although it is stereotyped as a thread which actually is essential, the Flyweight pattern helps for this thread to use efficiently the String object which results in a much efficient process.

The following diagram illustrates the design of the Flyweight pattern. Perhaps in this project its implementation has not been such object oriented, or without the Factory Pattern to retrieve the object, but the practical effects of Flyweight pattern are noticed boldly through the use of the Editor Component which uses Hyponym detection as results are shown quickly due to its application.



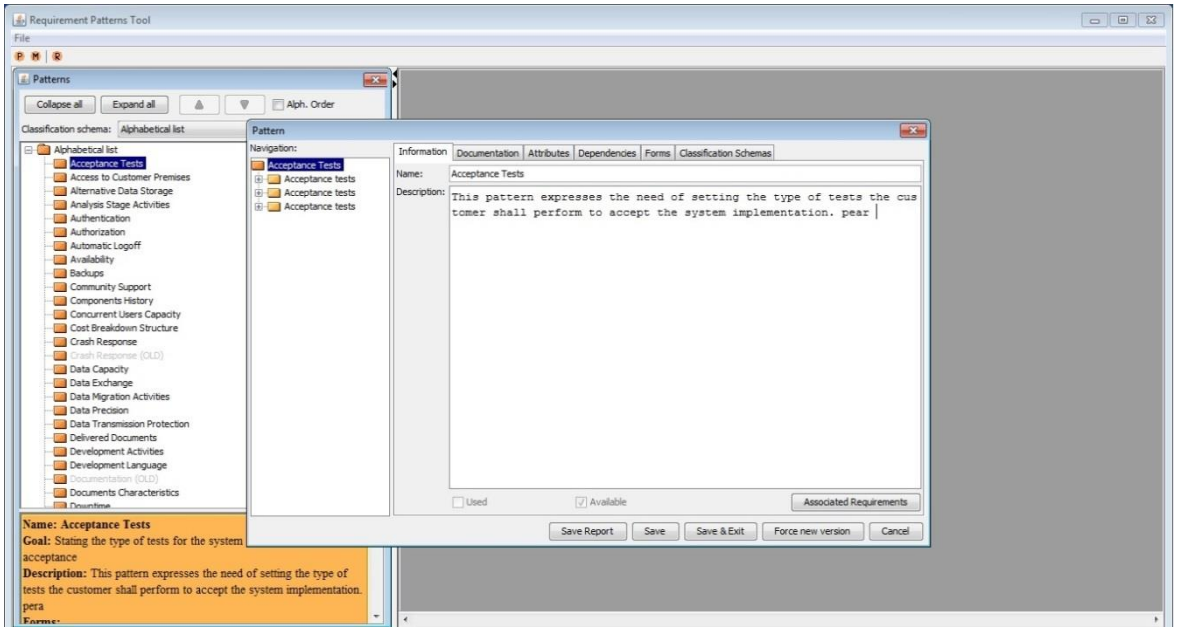
7 External Design

This chapter plays an important role in this project. Once the Business and Data Tiers are defined, the Presentation Tier needs to be designed in order for the extension in PABRE-Man to be synchronized in aspect and usability with the main application.

Conceptually, and in terms of presentation entities, this project introduces these main innovations to the tool [12] [SWN]:

- A customized input Text Component where Hyponym detection can be enabled and viewed. The total number of these Components must be situated in the correct way, size, and appearance, in the proper container as it was before the extension of the tool.
- A Glossary Internal Frame in which to view the contents of the Glossary in a disposed tree-like structure, attending user actions on nodes as well as a graphic management of the structure.
- A custom dialog to add or edit hypernyms and delete a quantity of the hypernym's hyponyms.
- Toolbar icon button to show or hide Glossary Internal Frame.
- Main Menu options to save to or import glossary from XML file.

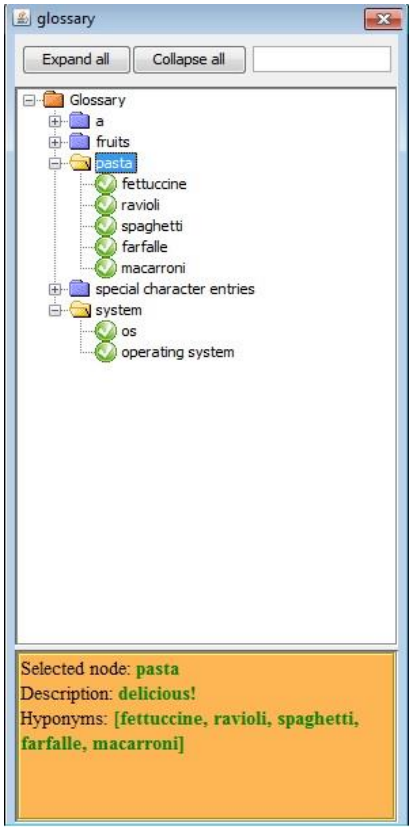
To have a visual reference of what the tool looked and felt before extending it, the following screenshot was taken:



7.1 Glossary Internal Frame

The main internal frame window where data from the Glossary is shown as well as were the user can manage it is the Glossary Internal Frame. Glossary Internal Frame is a realization of the tool's Generic Internal Frame from which the Metrics Internal Frame is also realized. With the presence of two, or more, internal frames, a basic window manager has to be set up. These internal frames are embedded inside a Main Frame which takes the main part of the application window space and where the Patterns Internal Frame has a fixed and permanent position.

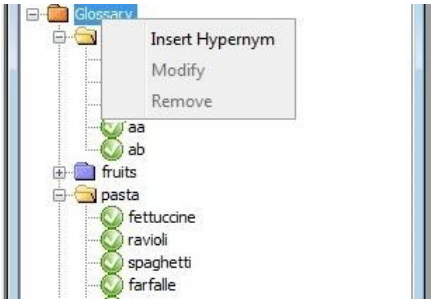
In order to render data from the glossary, this frame renders it with a Tree-like structure having a static node called "Glossary" as the root node, hypernyms on the first level nodes and hyponyms on the leave nodes. Focusing on this point, data received by the presentation tier controller has been rearranged to fit in this structure. This is the motivation of having a Multimap in the Business Tier Glossary object, in order to make a granularity leap to receive optimized data in the Presentation Tier, that is, for each hypernym a list of its hyponyms. As a result the following design is obtained.



The top part of the frame shows two buttons and an input text field. The buttons "Expand all" and "Collapse all "represent the visual functionality of expanding all hypernym nodes or collapsing them. For instance, in the figure, "pasta" node is expanded and "fruits" node is collapsed. "Collapse all" does not collapse further enough to the Glossary node, although it can be collapsed manually clicking on the -/+ button in the Tree-structure. This Tree-structure is clickable both by the primary and the secondary mouse button. If clicked by the primary

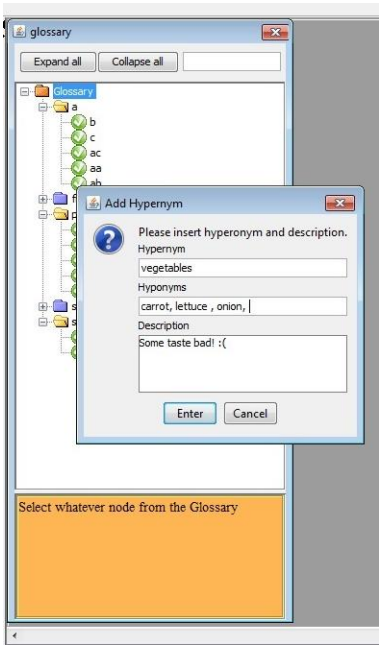
button the content of the selected node is shown in the lower bottom part of the frame - which can support Html coding- so if a hypernym is clicked the content shown is its description and hyponym list, and if a hyponym is selected its name is shown.

7.1.1 Insert hypernym



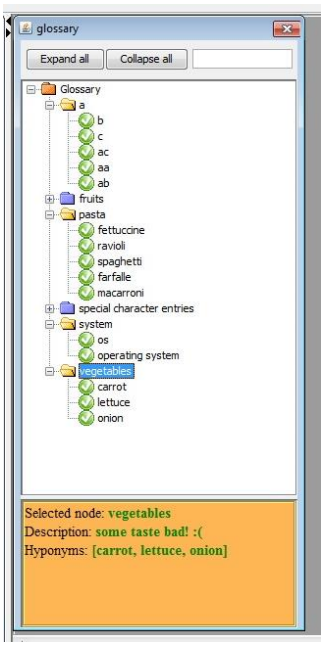
If the Tree-structure is clicked by the secondary mouse button it depends then on which node the user has clicked. If the user has clicked the Glossary node then only the insert hypernym menu is shown since it is the only option to be done from the Glossary node. Then this pop up menu is shown.

When the user confirms to insert a hypernym the Custom Dialog associated to the insertion of a Hypernym shows up. The user inserts the hypernym's name, and a list of hyponyms separated by commas, and a description for the hypernym.



If any of either the hypernym or one or more of the hyponyms does not pass the functional requirement's check an error message is shown.

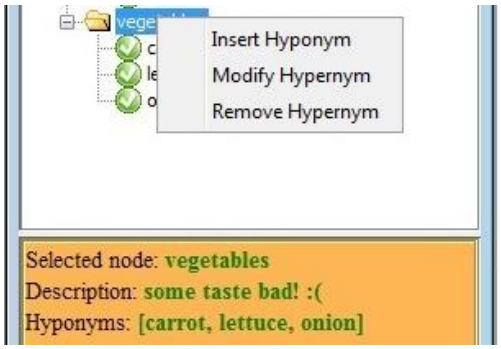
On success or on cancellation, the Custom Dialog disappears and the user is set back to the Glossary Internal Frame, with the inserted node selected.



Notice how nodes are redistributed alphabetically after the insertion, and that the previous opened and/or closed nodes are remained in the same state as before the insertion.

7.1.2 Hypernym menu

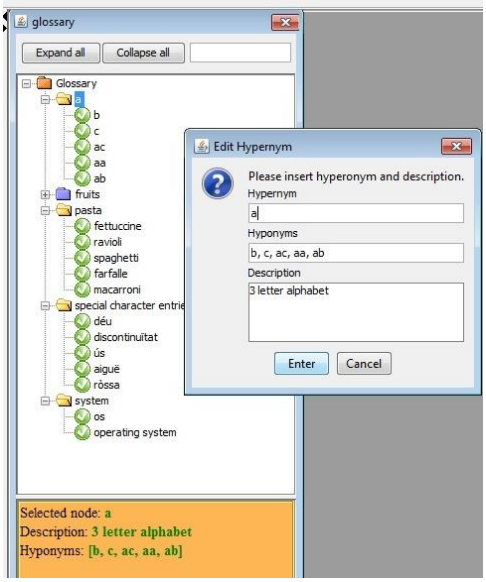
Again, the user can click the secondary mouse button on the hypernym node and watch the following popup menu with all options enabled:



The user also can edit a certain node (one at a time), either hypernym or hyponym, by double clicking it with the primary mouse button and modifying it. The user can cancel this edition action by pressing the Esc key or clicking anyway outside the Tree-like structure or confirm it by pressing the Enter or Intro key. If an edition is confirmed and violates the functional requirement's test, then an error message is shown and the old value of the node is reset.

7.1.3 Edit Hypernym

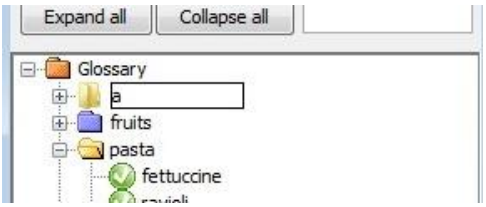
Focusing on editing a hypernym the first option is using the secondary mouse button option. This option leads to the same Custom Editor as in the insertion section, but with the fields already filled up of the data corresponding to the hypernym which is going to be edited.



This time the edited hypernym is the first one lexicographically, associating some of the three letter alphabet words with the letter (word) “a”.

The user shall introduce the modification and afterwards the modified data shall be, again, tested to pass the functional requirement test.

If so, the data will be modified accordingly. In this case, an interesting aspect arises. If the user erases a hyponym in the text list at the same time it modifies the hypernym’s name in a valid way, the hypernym’s hyponyms shall be modified so, erasing the missing values. However this case shall be discussed graphically in the hyponym’s modification section below.

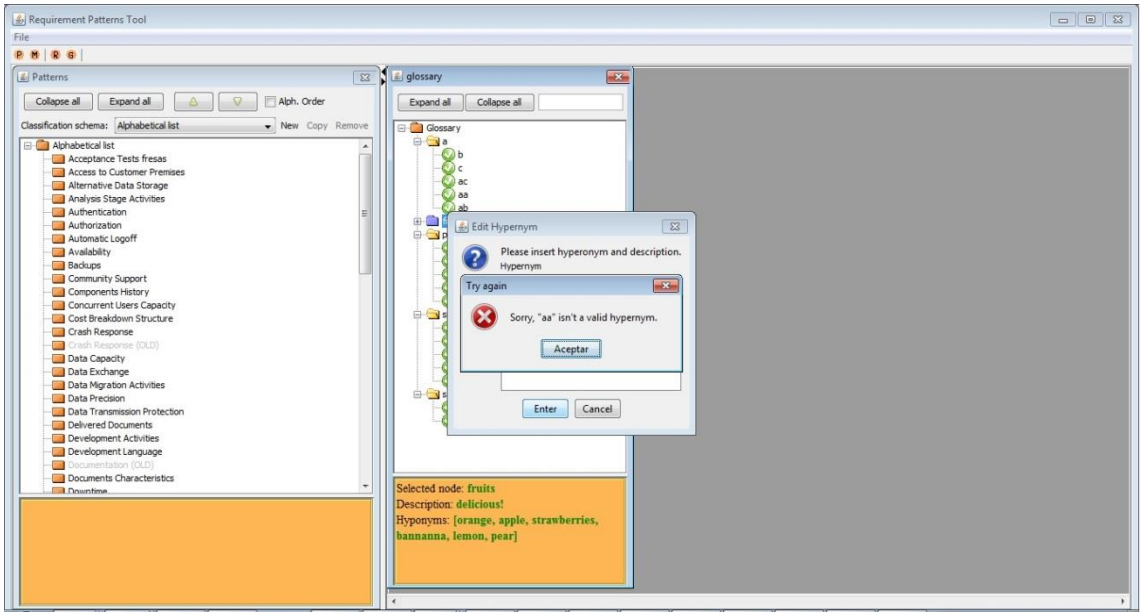


This is the same case but having the hypernym edited by double clicking its node with the primary mouse button. The node changes render form and node text becomes editable.

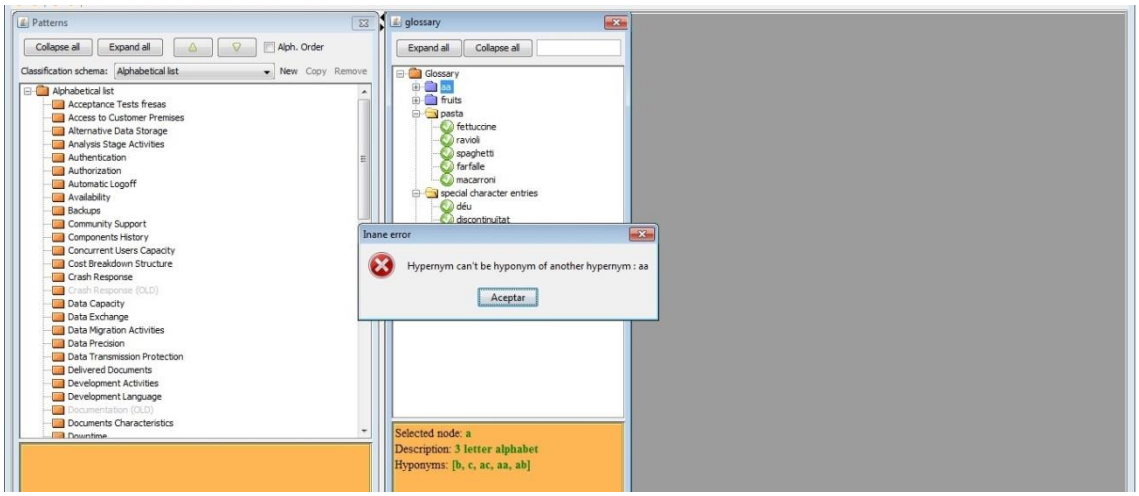
7.1.4 Error messages

For every insertion or modification there can be a violation of the functional requirement's test. For instance these are the ones addressed and based upon the above case.

In the case of editing the hypernym with the custom dialog this error is shown.



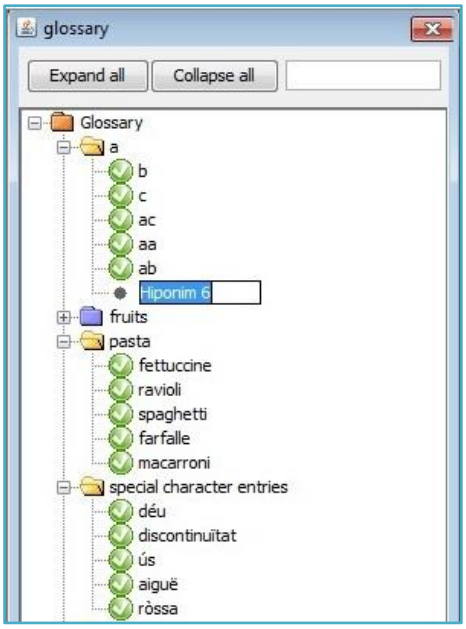
Notice the edit custom dialog does not disappear in the presence of the error message. Thus the system enables the user to give another trial or cancel the edition. In this case, the hypernym "fruits" was appointed to change its name to "aa", which is a hyponym from "a", so that's the reason why the message is shown.



The same situation, were the hypernym is being edited by double clicking the hypernym node.

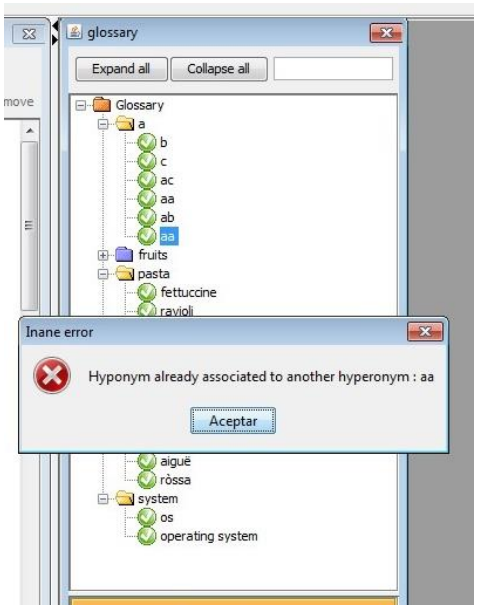
7.1.5 Insert hyponym

After selecting option “Insert hyponym” in hypernym node pop-up menu option, a new hyponym node is created as son of the selected hypernym with the name “hyponym X”, where X is the number of sons –including the just created one- the hypernym has. The new hyponym is selected and editable in order for the user to change its name.



If the user clicks on other place or cancels the edition of the new node by pressing Esc the node automatically disappears without being saved in any memory object or persisted object, that is, it only existed on the presentation Tier.

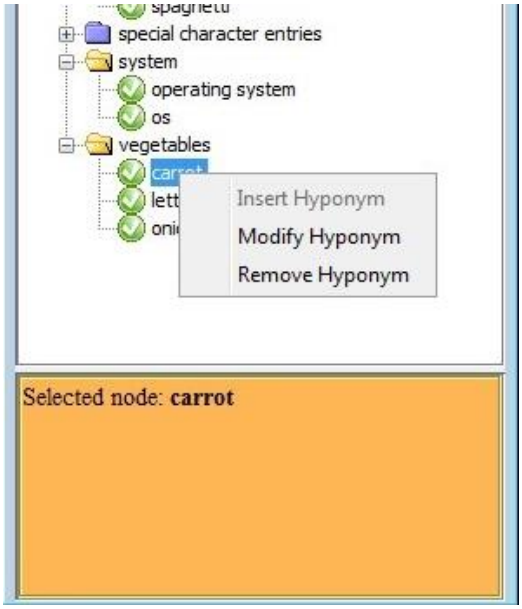
If the user confirms the edition of the new node, then the functional requirement's test has to be checked. Upon success, the hyponym is added in the implicit Business Tier object.



Upon error, the consequent error message is shown.

7.1.6 Hyponym menu

When clicking with the secondary button of the mouse in a hyponym node the options that the user will see are the following.



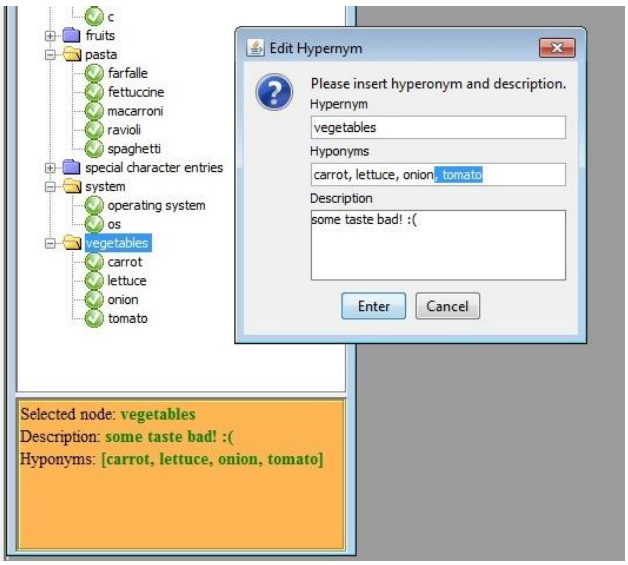
Insert Hyponym must be disabled so the “one level hierarchy” functional requirement restriction is fulfilled also in the Presentation Tier.

The remove hyponym option will simply delete the node in the Tree structure, as well than removing it from the Business Tier object.

7.1.7 Hyponym modification

When the option of the hyponym pop menu “Modify Hyponym” is selected the selected hyponym is set editable for the user to modify the hyponym and the same final process as in hyponym insertion starts. If the modification satisfies the functional requirements test then the value is pushed down through the controllers to the Business Tier object. If the modification does not satisfy the requirements an error message is shown the node’s value returns to the old pre-modification value. The same happens if cancelation pushing Esc key or clicking elsewhere in the frame takes place. The node is not deleted in this process.

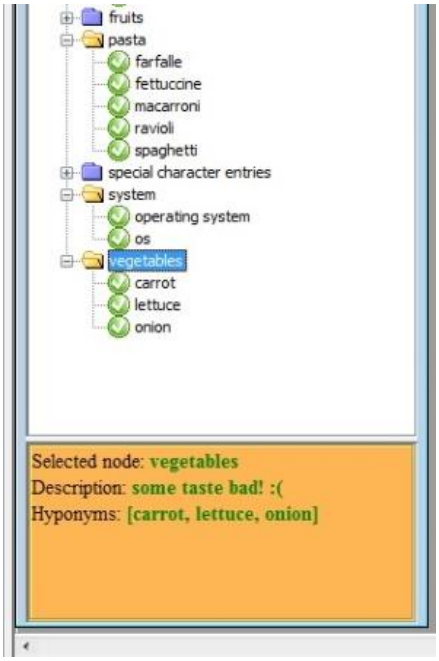
As mentioned in the hypernym modification section some pages above, there can also be a case of hyponym deletion when modifying hypernyms. The example is as follows: An insertion of the word tomato is associated to the hypernym vegetable (a tomato is a fruit). The user wants to correct this error and uses the edition custom dialog to specify the deletion implicitly instead of clicking the tomato hyponym with the secondary mouse button and clicking on the Remove Hyponym option which is also allowed and functional.



It is clear how the next step is erasing the selected content and clicking Enter button.

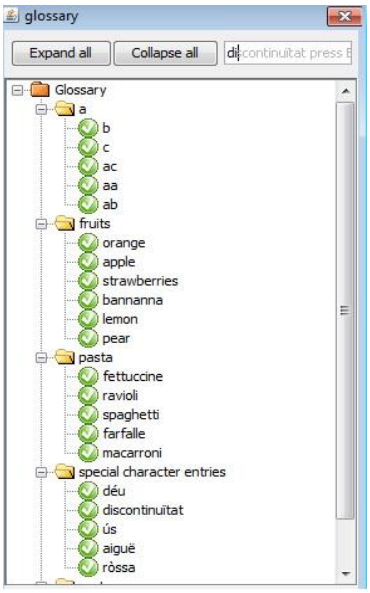
It is not possible to show an error message to this method because all the hyponyms listed, including the candidate/s to be erased, were already checked in the insertion process.

After the Enter button this is the result of the Glossary Internal Frame. The tomato hyponym has been erased from the Tree-Structure and from the Business Tier object.



7.1.8 Search Functionality

In order to search for a certain node within the Glossary Internal Frame Tree Structure, an auto-complete input searcher has been implemented. It is especially useful if the size of the glossary increments to dimensions where searching nodes visually becomes certainly difficult.

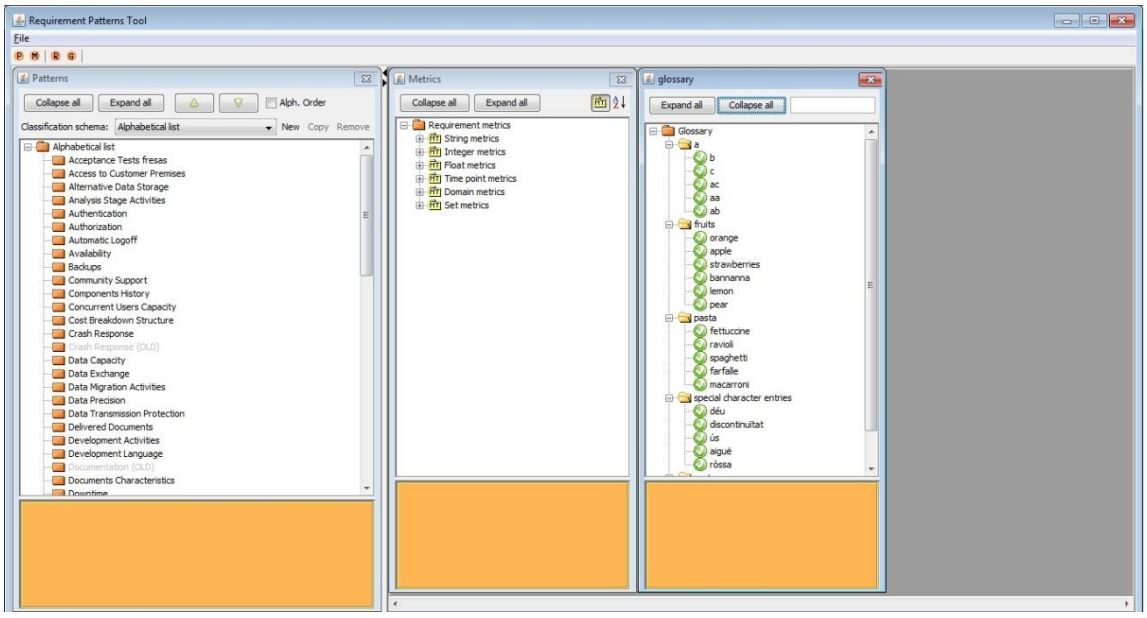


This functionality easies even further the search suggesting the most probable lexicographical word whilst the user is still typing its prefix.

When the user finds that the searched word is suggested in the search input text field he/she just has to press the Enter/Intro key and node of the searched and confirmed word will be selected.

7.1.9 General External Appearance

The incorporation of the Glossary Internal frame with the entire application results as shown in this screen capture.

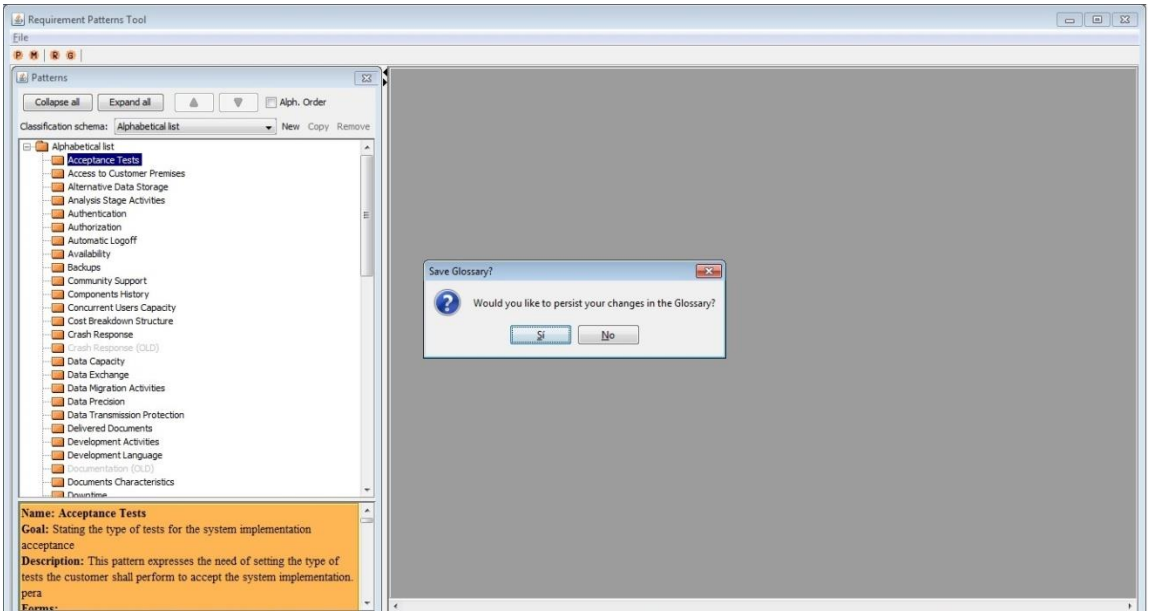


Glossary Internal Frame has been open in second position. If it were opened on first position, Metrics Internal Frame and Glossary Internal Frame would have swapped positions. A position manager has been implemented to achieve this same effect if a third Internal Frame were to be added and its position to be managed.

7.1.10 Glossary Internal Frame Persistence Dialog

If the Glossary Internal Frame is closed, there is a prompt which asks the user if he/she wants to save the glossary or not. In other words, if the implicit Business Tier object is wanted to be saved into the database system through the Hibernate enabled functionality. If the answer is yes, the Business Tier Glossary is saved, or updates an older version, in the Database system. If the no option gets selected, then the Business Tier Glossary is told to evict himself, which is in other words, to flush the intermediate changes and to acquire the database data in order to be synchronized correctly with strictly valid data.

The following dialog is shown when closing Glossary Internal Frame.



Notice the Glossary Internal Frame window is already closed and the main window is prepared but blocked until the dialog is answered.

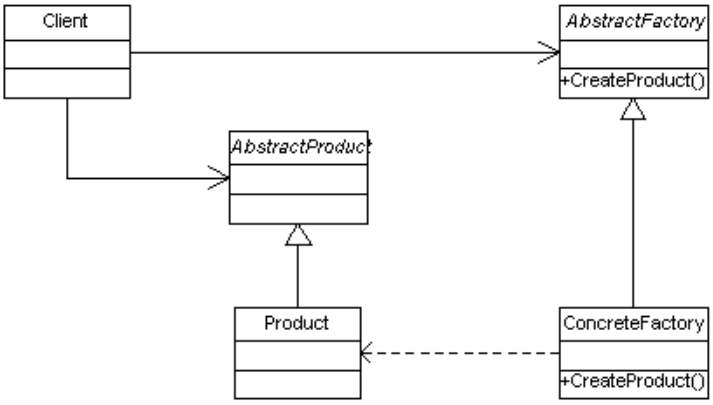
7.2 Editor Component

Auto Hyponym Check and word substitution by means of a pop-up menu handler showing after clicking with the secondary mouse button cannot be made possible using Java standard input text components. Some components must be extended and their methods overridden to achieve the goal of obtaining the desired component. However, some circumstances will apply the need to insert a single line input text field, others a larger and scrollable text area, and others a text area which follows the caret position and does not word wrap the text to a new line.

Besides this, among all this components sometimes a character limit has to be set in order for the user not to exceed it. If the user wants to type the char-limit character, a system beep is shown and that character is not added to the prompt input text.

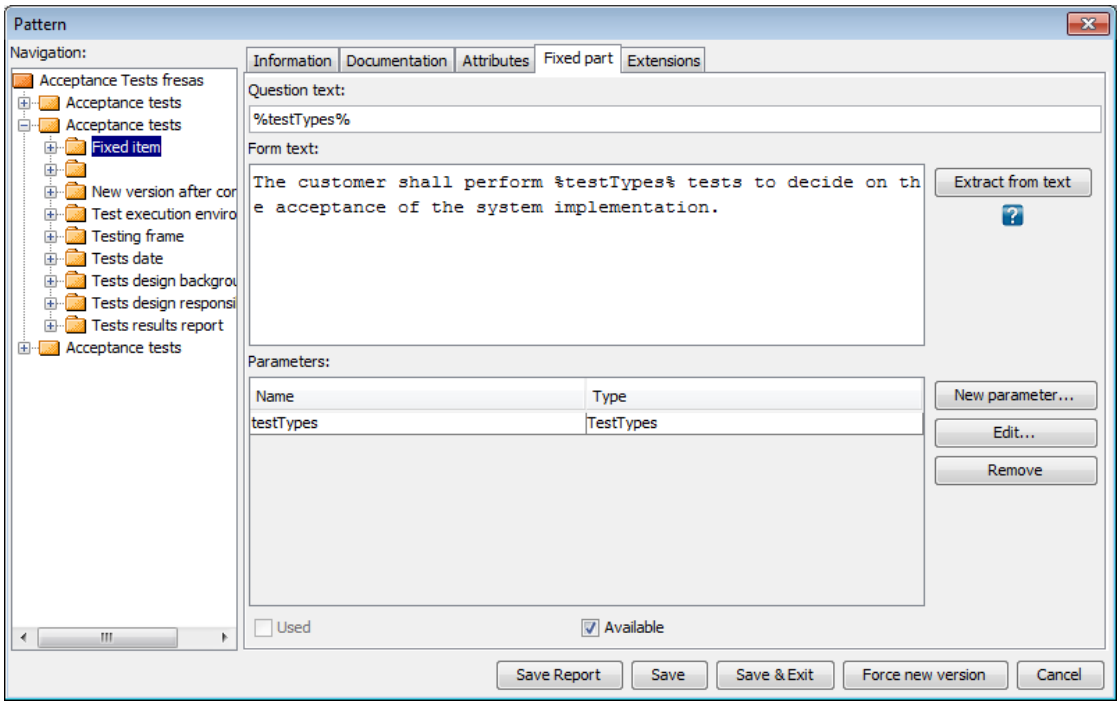
With these considerations the Abstract Factory Pattern fits like a glove for this situation. Additionally, since all components are susceptible to be added into a layout or other component, a hook method to get the actual component among all the possible ones has to be set to this Factory of objects. Again, all of the components are likely to have a setter and getter methods which manage their text inputs. This is why also the Proxy Pattern –already seen in the design section- is also is applicable.

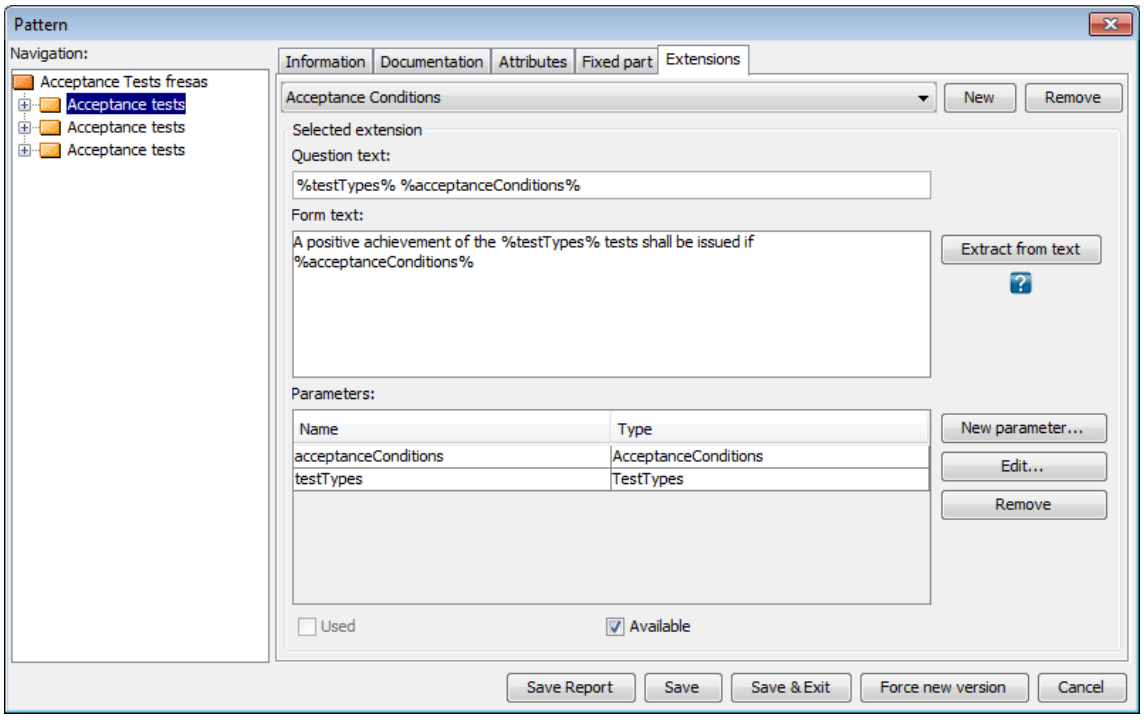
The Abstract Factory Pattern is shown below:



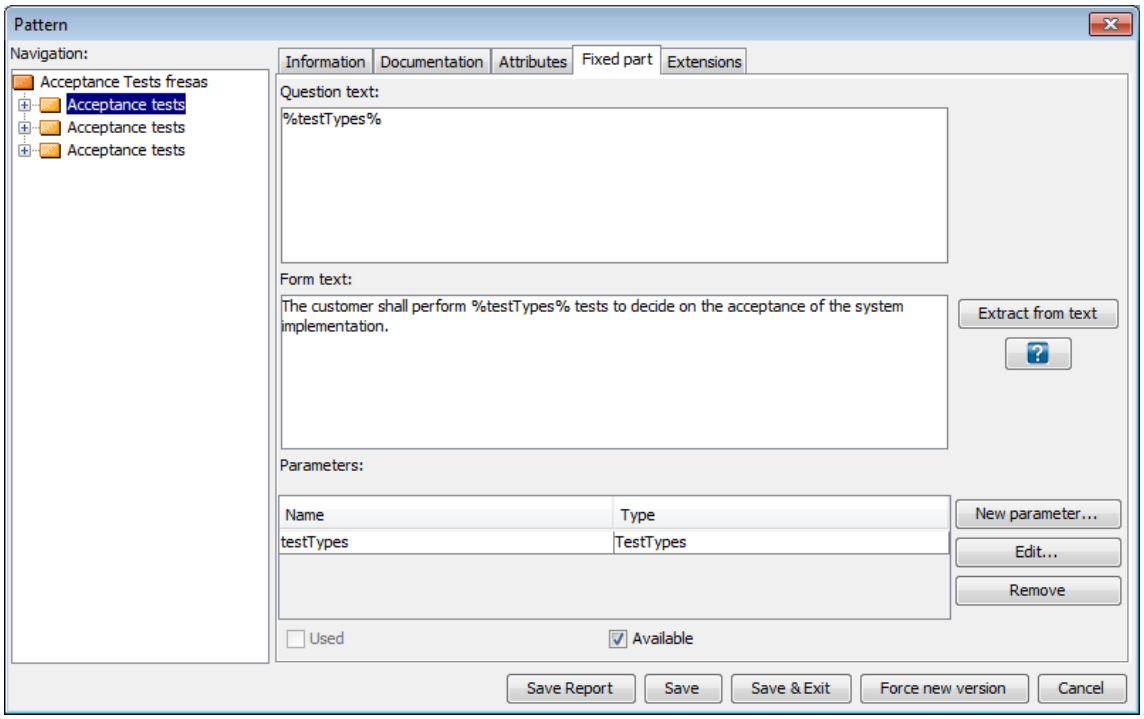
This new Editor Component has been added to various components in the PABRE-Man tool. For instance, these are some screen shots to illustrate the before and after effect of this addition in the most relevant features where it is applied.

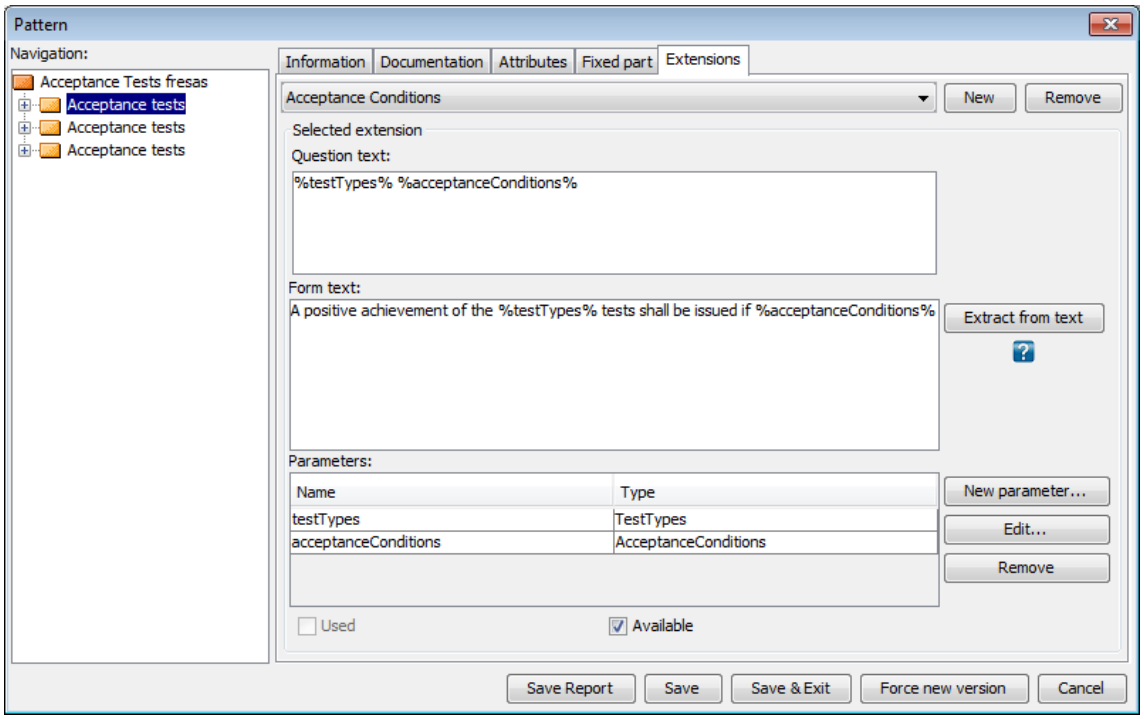
Before


Pattern
Navigation:
Acceptance Tests fresas
Acceptance tests
Acceptance tests
Fixed item
New version after cor
Test execution enviro
Testing frame
Tests date
Tests design backgrou
Tests design responsi
Tests results report
Acceptance tests
Information
Documentation
Attributes
Fixed part
Extensions
Question text:
%testTypes%
Form text:
The customer shall perform %testTypes% tests to decide on th
e acceptance of the system implementation.
Extract from text
Parameters:
Name
Type
testTypes
TestTypes
New parameter...
Edit...
Remove
Used
Available
Save Report
Save
Save & Exit
Force new version
Cancel


Pattern
Navigation:
Acceptance Tests fresas
Acceptance tests
Acceptance tests
Acceptance tests
Information
Documentation
Attributes
Fixed part
Extensions
Acceptance Conditions
New
Remove
Selected extension
Question text:
%testTypes% %acceptanceConditions%
Form text:
A positive achievement of the %testTypes% tests shall be issued if
%acceptanceConditions%
Extract from text
Parameters:
Name
Type
acceptanceConditions
AcceptanceConditions
testTypes
TestTypes
New parameter...
Edit...
Remove
Used
Available
Save Report
Save
Save & Exit
Force new version
Cancel

After





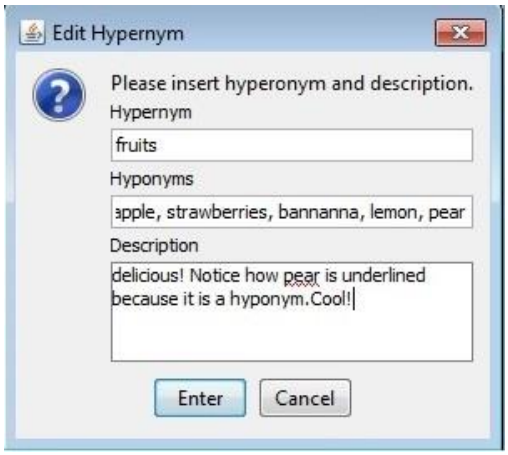
Notice how both Question (very clearly) and Form texts have become larger in size. These both input text will be retrieving invalid words by highlighting them at the time the invalid words are written, and support the event that will trigger their substitution.

7.3 AutoHyponymChecker

This section corresponds to the Presentation Tier action of highlighting the invalid words, or hyponyms, in order to make the user clear there has been an invalid word localized in the written, loaded or pasted text.

There are several locations were invalid words can be searched for and others locations where no searching has to be made. For instance in the Description section of the hypernym's Custom Dialog, either in inserting or editing mode, this checker has to be enabled.

This is why in the following screenshot an invalid word is located.



This way the user can choose either to correct the invalid word by a hypernym or to leave it this way.

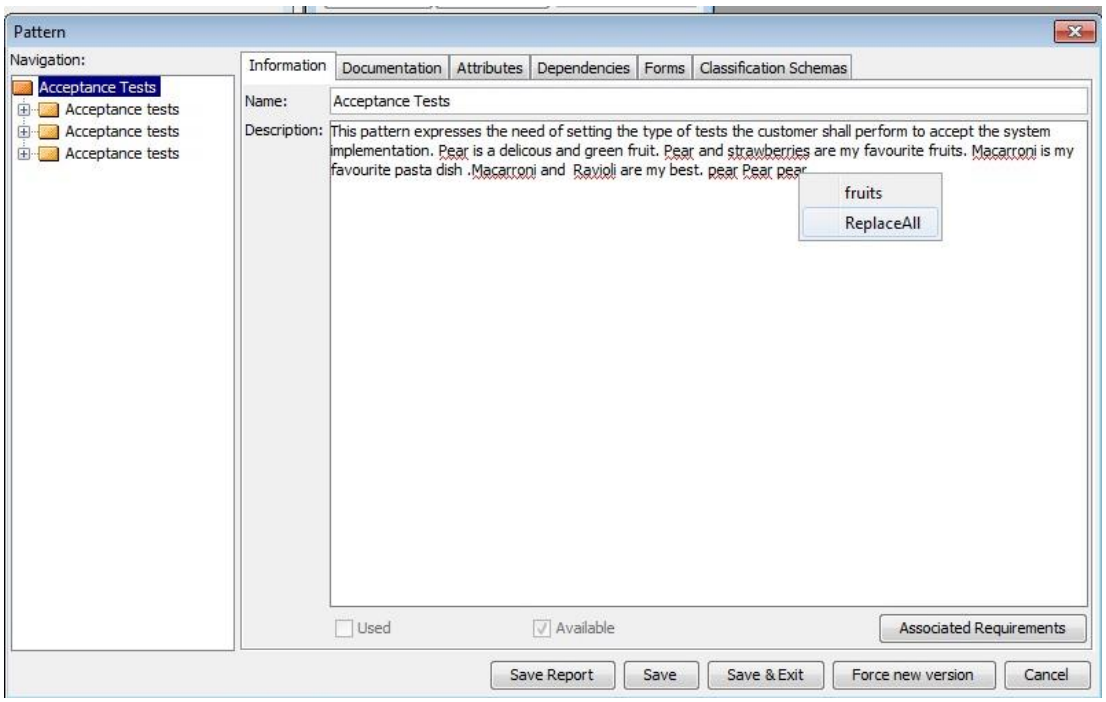
Other places suitable to find invalid word highlighting are Pattern forms. In certain fields such as Name, Description, attributes' Goal and Keyword, Fixed and Extended part's Pattern Text and Question Text highlighting shall be enabled. In metrics creation or modification form, the String metric's Default value shall be also checked as well as the input field corresponding to adding a new Domain Metric. Other input dialogs showing plain messages are also susceptible to show highlighting.

7.4 Hyponym Substitution

Once the user has decided to replace a hyponym which has been highlighted, he/she clicks on top of the desired word with the secondary mouse button a pop menu showing the hyponym's hypernym and a constant message "Replace All" is shown. If for somehow the user clicks either with the primary mouse button or with the secondary mouse button in any other position not coinciding with a hyponym, nothing is to be done -an exception trace is not to be shown- and the application flux must not even notice interruptions.

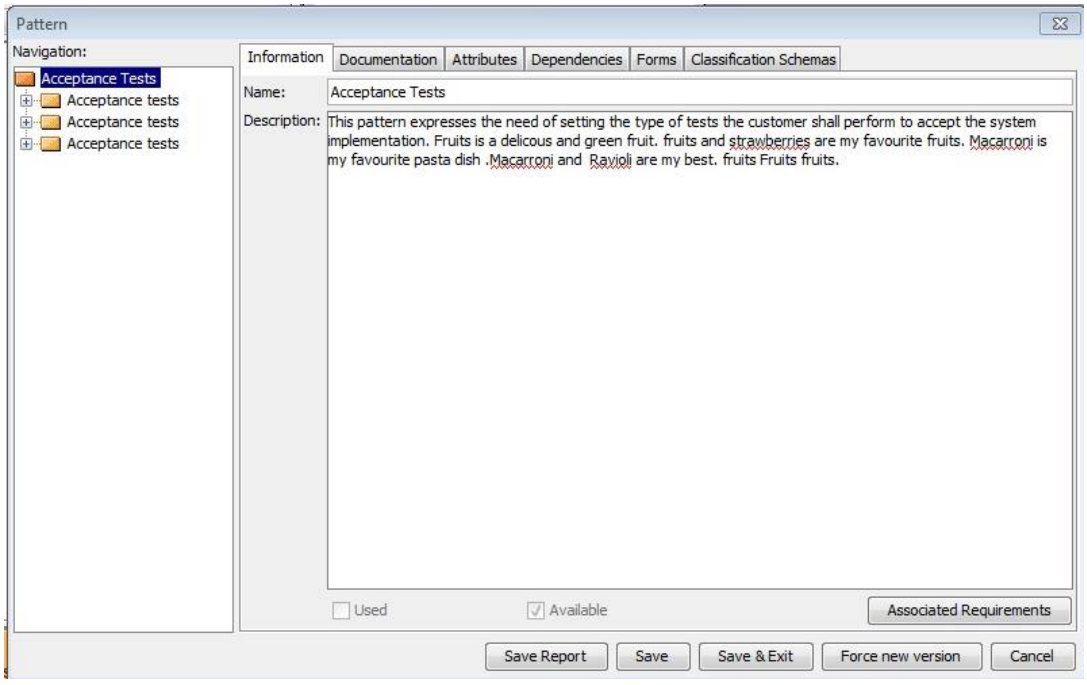
This is the screenshot storyboard of a “Replace All” substitution.

Initial context:

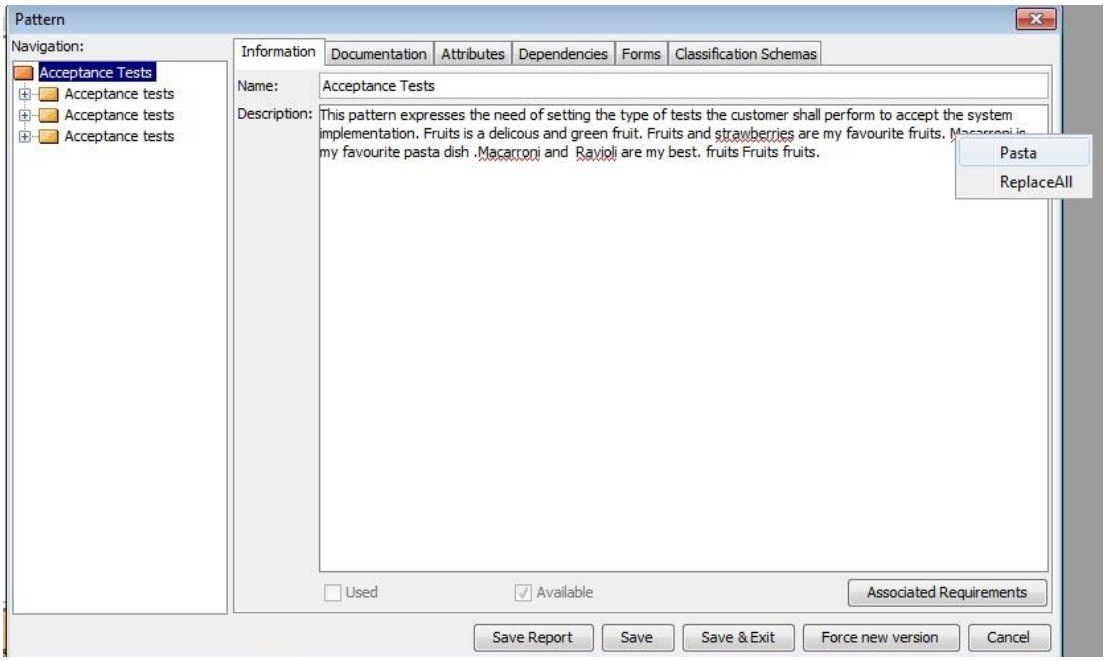


Notice the user has clicked over the last pear word, and the next step is to click on the “Replace All” sub-menu so all the pear word occurrences will be replaced by the word “fruits”. Behold also the capitalization of the hyponym words which shall be mocked by the hypernym word.

Replace all substitution:

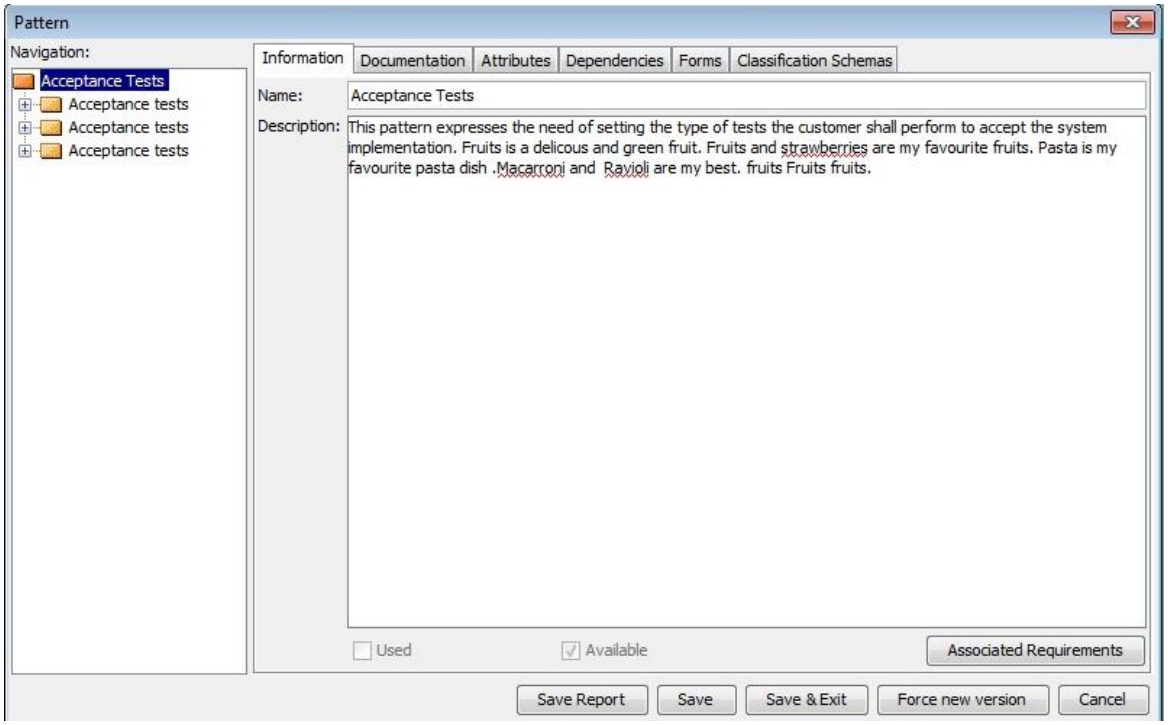


Now, a single substitution is to be made. The initial context is now set after the user has replaced al pear occurrences.



Notice how the hypernym option is given with the first letter capitalized because the hyponym word was also capitalized because it was the first word in the sentence.

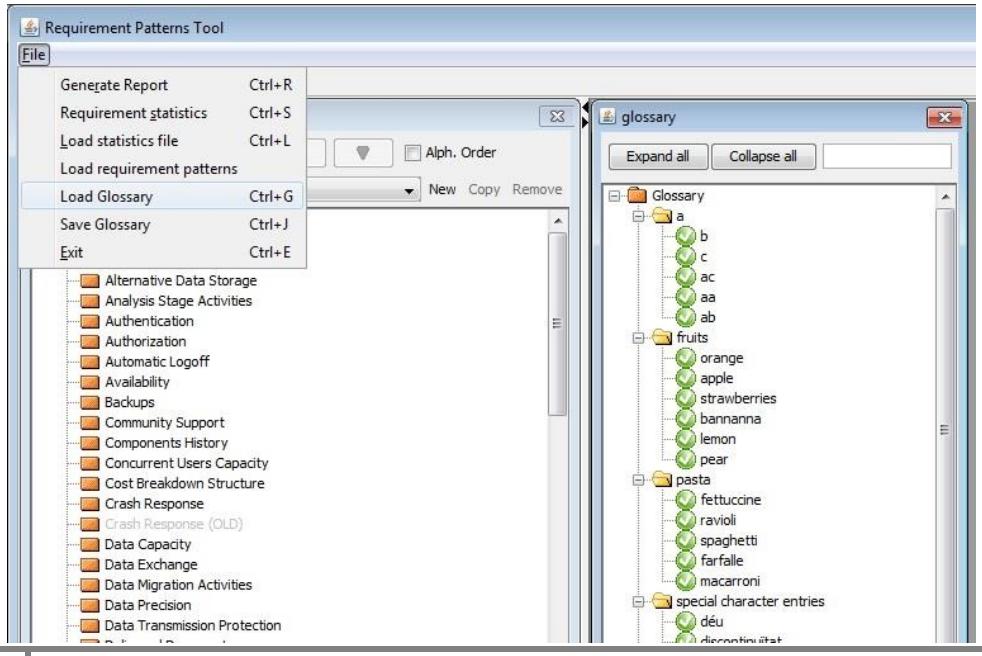
This is, then, the result of the single substitution:



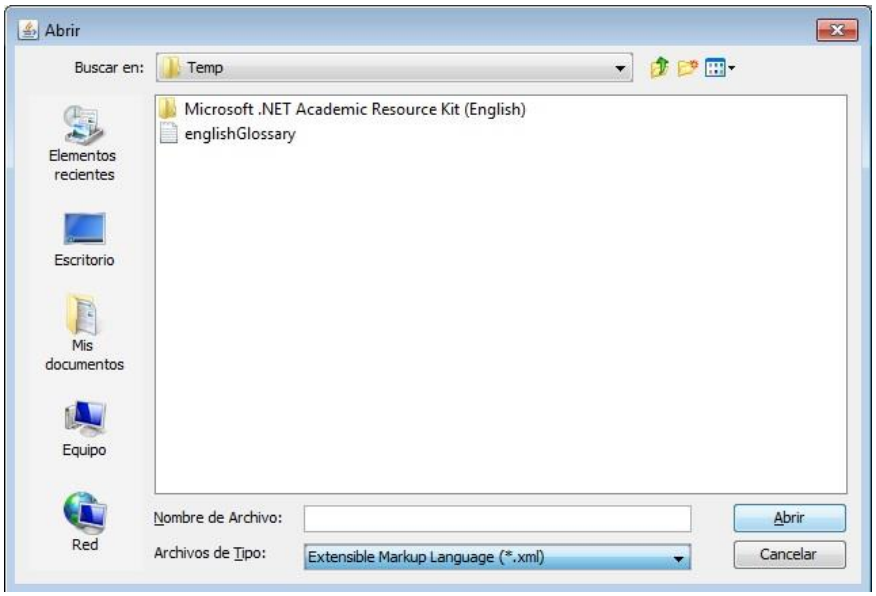
User can then interact with the resulting data as before the extension of the PABRE-Man tool.

7.5 Importation / Exportation of the Glossary

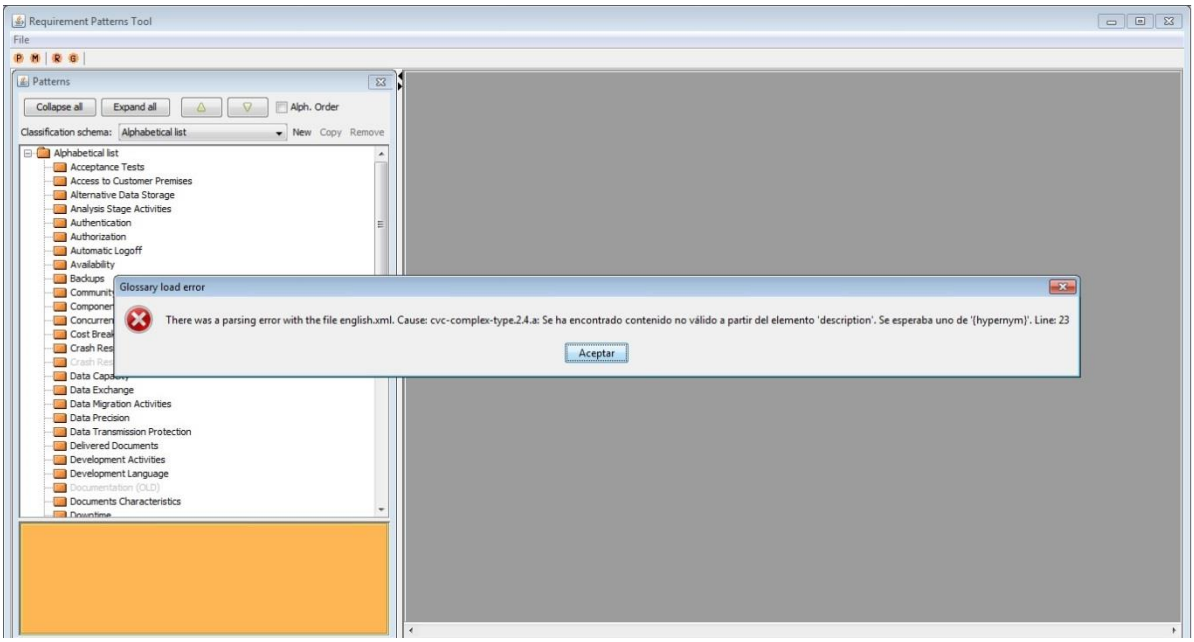
In order to speed up the process of Glossary importation and exportation, a menu option is added to the main menu option bar. A keyboard shortcut is also operative: Ctrl + G to load the Glossary and Ctrl +J to save it.



In order to open the XML file containing the Glossary data, the typical Open file dialog with XML file filter is shown for the user to select the desired file and thus for the system to know the file path.



The only error associated to importation or exportation of the glossary is the importation of a badly formed XML file. In that case, the following error message is shown and the Glossary is not loaded.



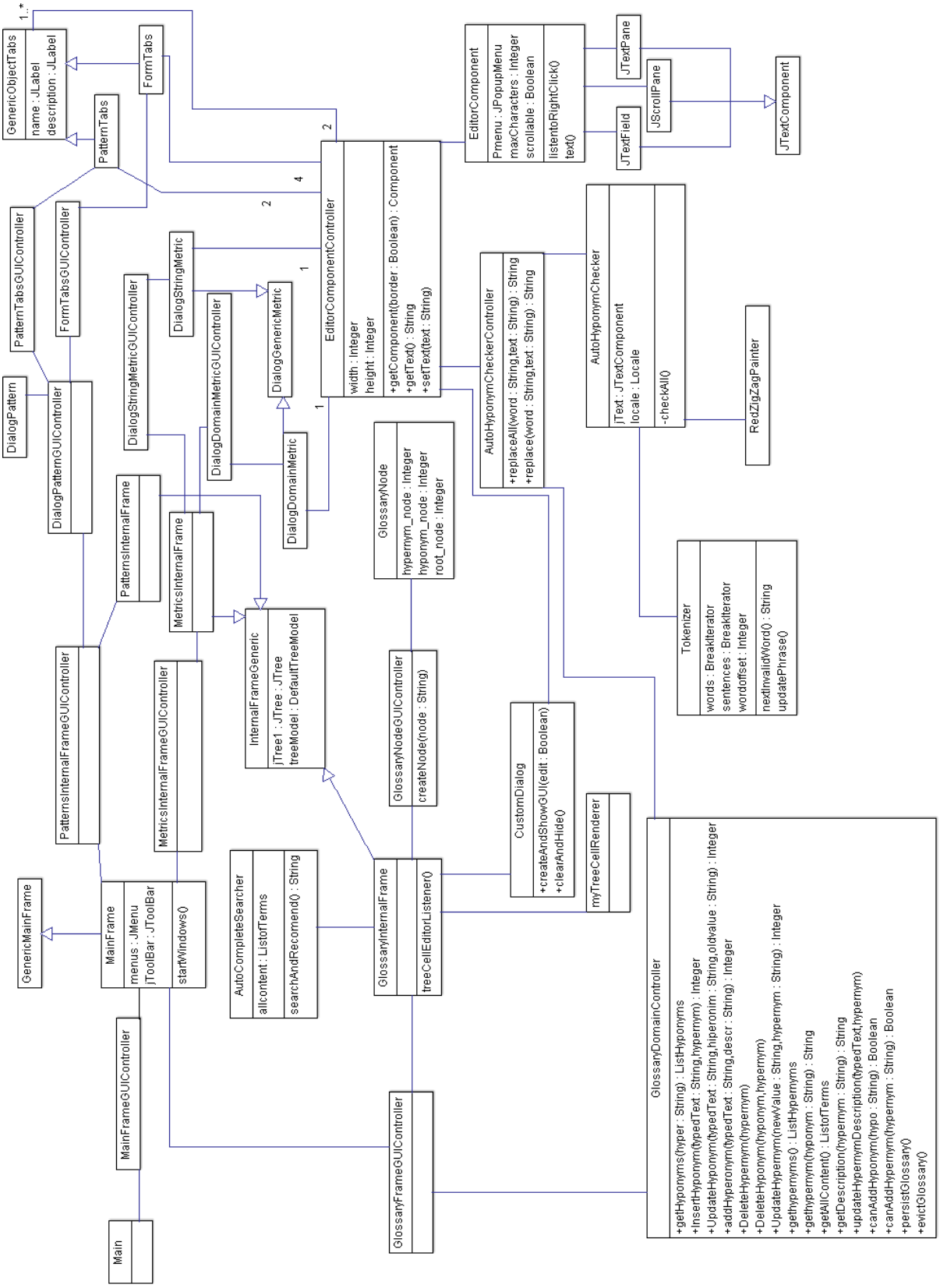
It is remarkable how the own error message identifies the parsing error in the XML file and gives the user valuable feedback to fix it.

7.6 Design of the Presentation Tier

A general portrait of what is happening in the Presentation Tier escapes from the scope of this project. However it helps visualize how the different views, frames and visual functionalities interact and relate themselves in the PABRE-Man tool after the extension for which this project is responsible is installed. Certainly this Tier is really dependent on the technology as for how technology fetches and treats the different events the user makes or the functionalities he/she demands. With all, this is a brief design of the whole Presentation Tier.

By finishing this chapter the following initial sub-objectives are fulfilled:

- **Integration**
- **Usability**


GenericObjectTabs
name : JLabel
description : JLabel
FormTabs
PatternTabs
EditorComponent
Pmenu : JPopupMenu
maxCharacters : Integer
scrollable : Boolean
listentoRightClick()
text()
JTextPane
JScrollPane
JTextField
JTextComponent
PatternTabsGUIController
FormTabsGUIController
DialogStringMetric
DialogStringMetricGUIController
DialogGenericMetric
DialogPattern
DialogPatternGUIController
DialogDomainMetricGUIController
DialogDomainMetric
EditorComponentController
width : Integer
height : Integer
+getComponent(border : Boolean) : Component
+getText() : String
+setText(text : String)
AutoHyponymChecker
jText : JTextComponent
locale : Locale
-checkAll()
AutoHyponymCheckerController
+replaceAll(word : String,text : String) : String
+replace(word : String,text : String) : String
RedZigZagPainter
PatternsInternalFrame
MetricsInternalFrame
GlossaryNode
hypernym_node : Integer
hyponym_node : Integer
root_node : Integer
InternalFrameGeneric
jTree1 : JTree : JTree
treeModel : DefaultTreeModel
PatternsInternalFrameGUIController
MetricsInternalFrameGUIController
GlossaryNodeGUIController
createNode(node : String)
Tokenizer
words : BreakIterator
sentences : BreakIterator
wordoffset : Integer
nextInvalidWord() : String
updatePhrase()
CustomDialog
+createAndShowGUI(edit : Boolean)
+clearAndHide()
myTreeCellRenderer
GenericMainFrame
MainFrame
menus : JMenu
jToolBar : JToolBar
startWindows()
AutoCompleteSearcher
allcontent : ListofTerms
searchAndRecomend() : String
GlossaryInternalFrame
treeCellEditorListener()
MainFrameGUIController
GlossaryFrameGUIController
Main
GlossaryDomainController
+getHyponyms(hyper : String) : ListHyponyms
+InsertHyponym(typedText : String,hypernym) : Integer
+UpdateHyponym(typedText : String,hiperonim : String,oldvalue : String) : Integer
+addHyperonym(typedText : String,descr : String) : Integer
+DeleteHypernym(hypernym)
+DeleteHyponym(hyponym,hypernym)
+UpdateHypernym(newValue : String,hypernym : String) : Integer
+gethypernymsS() : ListHypernyms
+gethypernym(hyponym : String) : String
+getAllContent() : ListofTerms
+getDescription(hypernym : String) : String
+updateHypernymDescription(typedText,hypernym)
+canAddHyponym(hypo : String) : Boolean
+canAddHypernym(hypernym : String) : Boolean
+persistGlossary()
+evictGlossary()

8 Work plan

It is fundamental to manage time and resources in nowadays projects. For this reason methodologies have been set to control and monitor the projects processes and evolution in function to an initial kickoff.

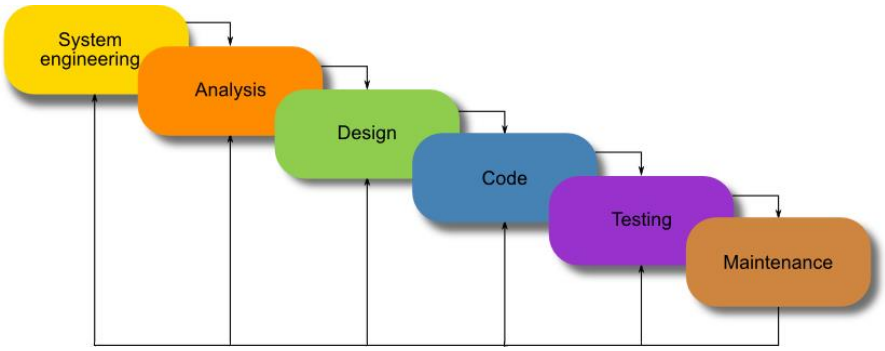
As a resume, this project starts with a stable version of the PABRE-Man system, and its purpose is to append a software system to it in order to extend it. The glossary is dependent with the system only in the terms the PABRE-Man tool uses. For any other aspect it seemed to be totally independent. However it turned out for integration of the glossary interface to the presentation layer to be a cost full task.

First of all a methodology must be selected to be followed through the development of the project. Next, an extensive initial work plan has to be presented with an exhaustive task breakdown. Afterwards, this initial work plan has to be compared and contrasted with the real work plan, due to the fact that it is probable that tasks can have been taken more time than predicted. The real work plan has to be used to calculate the final economic cost of the project.

8.1 Methodology

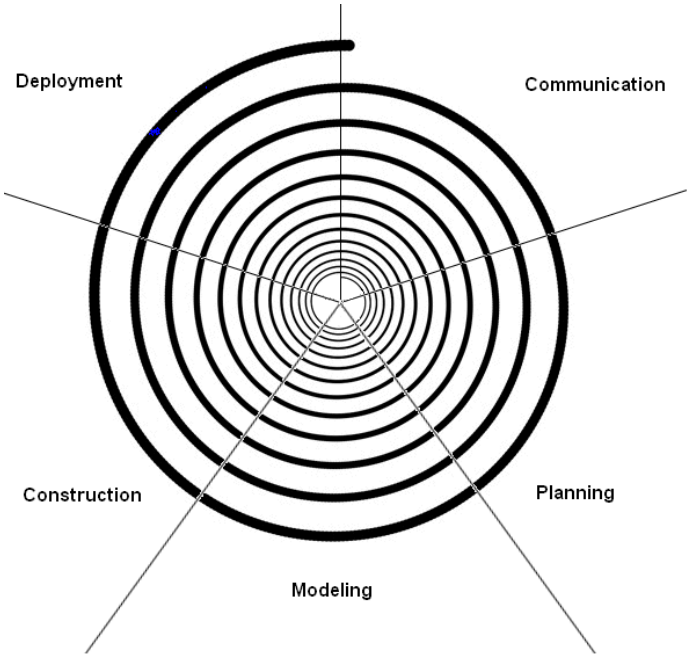
8.1.1 Introduction

There are several methodologies to use in order to develop quality software. For instance, let's consider the classical life-cycle software model as a reference.



In this project's case Systems engineering is quite quick and simple due to the concreteness of the project and the sharp requirements. Where much labor has to be shown is in the design process, as well as in the coding or integration process. Once the analysis process is done, there are tasks within the project which can be done concurrently, without any order preference. Therefore this classical model is a reference in the general perspective but does not adapt to the actual needs of this project.

Another traditional model methodology is the Spiral Prototype model. It basically centers its issues upon general objectives which, as the process is iterated in spiral, are refined to optimize the quality of software.



Essentially, both traditional models are proper in the semantics of the methodology because the goals and processes are similar. However they are conceived to lead the building of Software Information Systems –many times from scratch- through a very tight, formal and procedural way leaning on the advantage it gives to tidy all parts of the software up in a clean manner. Besides, a sequential order is imposed, no other step can be made before the previous one has ended, and that can really be a bottleneck in the throughput.

Searching for quick and efficient development Agile Methodologies were created. They are based on iterative and incremental development, where requirements and solutions evolve through collaboration between developers and product owners. Agile development relies on adaptation more than to rigid specification and obeys the following principles:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan
- Regular adaptation to changing circumstances
- Face-to-face conversation is the best form of communication
- Welcome changing requirements, even late in development

8.1.2 Selected Methodology

Within Agile Methodologies Scrum is one of the most popular. Scrum is iterative, incremental and agile so it collects bits of Spiral Prototype and the classical life-cycle software model.

There are several roles to apply in this methodology:

- Product Owner which represents the stakeholders and the customer's expectations
- Development team which is responsible of producing and developing the artifacts and application. Usually built up of 3-9 individuals with cohesive roles among them.
- Scrum Master is responsible for the proper functioning of the Scrum process. It usually is a person with leadership and point of contact between the other different roles.

Scrum's basic unit of development is the sprint. A sprint is a restricted to a specific duration and effort, typically two weeks. Each sprint starts with a meeting were tasks for the sprint are identified. On the meeting for next sprint or end-of-spring-review meeting progress of the previous sprint is evaluated and reviewed.

This following graphic illustrates Scrum's processes:

In the Scrum methodology requirements and use cases are expressed as elements of the product backlog. The product backlog is a list of growing prioritized functional and non-functional requirements rated in function of the value the Product Owner has rated them with. However the order and rating can change during the development of the whole project so this list is adaptive to change. [11]

The application of Scrum in this project has to be conditioned to reality, there is not a great enterprise or Business behind the final product, or groups of development teams assigned to the project for communication to be extremely vital. However, the essence of Scrum is kept clear in proportion to the scope of the project.

The project's backlog includes the general requirements that are to be accomplished, that is, to extend PABRE-Man tool to enable hyponym highlighting and substitution. The way to accomplish with it could have varied throughout the timespan of the project due to Product Owner's preferences or technological handicaps. There has been set a weekly or two-week meeting schedule to have fluid communication with the Product Owner and Scrum Master. Risk management is optimized to a maximum project time delay of two weeks in which, if persisted, risk shall be converted to consensual solutions or the scope of the project committedly re-dimensioned.

Scrum fits like a glove to the management of these processes in this project. But, in order to have successful sprints, the backlog must be breakdown to simple tasks associated to sprints' backlogs in order to achieve the final goal.

8.2 Task Breakdown and Sprints

This project has been disaggregated in sprints by the following schema:

SPRINT NUMBER 1		
TASK	LENGTH (hrs)	TYPE OF TASK
Project proposal and meeting	2	admin
Project acceptance and registration	1	admin
Tribunal Assignation	1	admin
Initial requirement gathering and kickoff	2	admin
Brainstorm and research	10	development
Use case and requirement definition	16	development
Backlog elaboration	24	development

SPRINT NUMBER 2		
TASK	LENGTH (hrs)	TYPE OF TASK
Workspace installation	16	admin
PABRE-Man tool installation	8	admin
PABRE-Man tool knowledge	24	development
Sandbox creation	16	development
Business Tier specification	24	development

SPRINT NUMBER 3		
TASK	LENGTH (hrs)	TYPE OF TASK
Business Tier coding	32	development
Business Tier testing	16	development
Persistence methodology research	32	development
Business Tier controller coding	16	development

SPRINT NUMBER 4		
TASK	LENGTH (hrs)	TYPE OF TASK
Hibernate knowledge	40	development
Functional Requirement testing	10	development
Business Tier integration	14	development
Hibernate persistence design	16	development

SPRINT NUMBER 5		
TASK	LENGTH (hrs)	TYPE OF TASK
Hibernate persistence coding	32	development
Functionality testing with persistence	16	development
Persistence Load Test	8	development
Java presentation knowledge	24	development

SPRINT NUMBER 6		
TASK	LENGTH (hrs)	TYPE OF TASK
Tree render of data	16	development
Event management adding/editing/deleting	24	development
PABRE-Man tool presentation Tier code knowledge	32	development
Midterm inform elaboration	8	admin

SPRINT NUMBER 7		
TASK	LENGTH (hrs)	TYPE OF TASK
GlossaryInternalFrame creation	32	development
Event management adding/editing/deleting	16	development
AutoHyponymChecker	16	development
Editor Component Creation	16	development
Custom Dialog creation	16	development

SPRINT NUMBER 8		
TASK	LENGTH (hrs)	TYPE OF TASK
Tuning up functionalities & controllers	16	development
Evict glossary + hibernate	40	development
Tuning up Hibernate error bugs	16	development
Editor Component Adaptation and extension	16	development

SPRINT NUMBER 9		
TASK	LENGTH (hrs)	TYPE OF TASK
AutoHyponymChecker controller	24	development
Extending Editor Component Substitution Event	16	development
Analyzing Pattern and Metric presentation forms	24	development
Insert Editor Component to Pattern and Metric forms	24	development
Insert Editor Component to dialogs	16	development
Adapt customized sizes of editor Components	16	development

SPRINT NUMBER 10		
TASK	LENGTH (hrs)	TYPE OF TASK
Glossary XML exportation/importation	24	development
Adapt Toolbar and Internal frame position management	24	development
Correct latter bugs	24	development

SPRINT NUMBER 11		
TASK	LENGTH (hrs)	TYPE OF TASK
Appoint the Tribunal	2	admin
Project Report	50	admin
Project Lecture	2	admin

According to this breakdown only the last part of the final sprint is to be done. With a total disposure of 870 hours distributed between administration tasks – total of 92 hours- and development tasks – 778 hours-. It is also remarkable to see how 84 developing hours were dedicated to learn new technologies and to land firmly and with deep knowledge in the project. In sprint number 6 there has been a total of 72 developing hours which did not contribute to a final efficient throughput because its work could not be integrated directly in the PABRE-Man tool due to architectonical dependencies and that had to be remade and reconsidered in sprint 7.

8.3 Project Cost

To dimension the cost of the effectuated work per hour estimation has to be made. In actual economic context the following roles can be used to make a statistical estimation of the average cost of a development job hour:

- Project Manager – approximately 60 €/hour
- Analyst Programmer (or Software Architect) –approximately 35 €/hour-
- Software Developer –approximately 25 €/hour-
- Training Grantee –approximately 10€/hour-

Acknowledging Scrum does not have these roles exhaustively typified, the tasks for which each of these roles is responsible for are to be done broadly. Perhaps all the development tasks would be made by a group of 3-4 persons exercising those roles' responsibilities, distributing the total distribution time and tasks among them. In cost perspective, the average of all the costs can be used to define an average developer responsible of doing all tasks and with a cost of approximately 35 €/hour. With a similar justification administration tasks shall have a 15€/hour price.

Total Project Cost
28,610 €

Giving a 1400 € margin for incidentals, estimating a total round cost of **30,000 €** seems accurate and reasonable.

With the alternative of deleting sprint 6 and minimizing the new technologies' learning curve based upon the scenario of a senior development the total number of development hours descends to 622 to add to the constant 92 administration hours. Definitively the price per hour also has to increase to a senior level development price, say +10 €/hour difference. The definitive grand total of the cost remains similarly in a 30 000 € range, specifically 29,370 €.

9 Test plan

For every information system testing is fundamental to know the correctness of its functionalities or the percentage of achievement of the functionalities in case of failure. Adding to this, in order to submit the project and according to Scrum methodology, all artifacts must be proven to work fully, since testing is a must in Scrum methodology. In the case of this project, every sprint has an incorporated timespan for testing the work the sprint has developed.

For example, for Sprint number 3, for task Business Tier coding there has been an important detached task of testing the code that has been created. This testing task is even reflected in the task breakdown because of the importance of the Business Tier code created, since it is the heart of the application.

Testing of every sprint code with an eye toward the integration of the sprint's code into the final system has made integration of code easier. In a higher level, the final integrated code must further be integrated to the PABRE-Man system, which also demands testing plans in order to evaluate the correctness of the integration process. It may seem to be doing twice the work decreases working speed output parameters, and in a certain way it does, but the increment of effectiveness in orders of integration just makes worth it.

The final integration process includes the final evaluation testing of every artifact in close contact with the integrated code, to evaluate if its performance is correct after the integration of the code or at least to check it behavior is exactly as it was before.

Translating this process to this project the most significant testing points shall be Editor Component class – controlled by its respective controller (Editor Component Controller) – as an interaction with the PABRE-Man system, and the management of the glossary both functionally and visually. Testing the correct interaction of the Editor Component with the PABRE-Man system involves checking if the behavior of the Editor Components is the same as the artifacts the Editor Components replaced.

For instance, as an example of the testing process, in some input text fields if the user somehow managed to introduce a void string, either by the key press event of the key that voided the string or by a focus change event, an error message is triggered showing the user that void strings are not allowed in that input text and introducing a constant text message in that specific input text field.

This is achieved by designing Editor Components to have the same behavior and methods as the previously used components. Analyzing previous components their outstanding characteristics were that they all used the same Java Swing Class (JTextField) and the same methods of getting and setting the text attribute of this class. Requirements stated there had to be different sizes and behaviors of components adding to the functionality of hyponym highlighting and popup creation with its hypernym– which initially JTextField does not have- .

The solution goes through extending a general class which can be specialized in several subclasses which share the same hook methods retrieving and setting the text attribute and retrieving the component in order to be set in the layout.

This component will be different among each different type of Editor Component but will fulfill the requirements of enabling the Editor Component to be set in the same layout of the previous JTextField, and continue to feed the functionalities which where associated to the events of the JTextField.

Testing the creation of Editor Component seeking for correctness is a must within the Scrum methodology in Sprint number 7. Without undergoing much in technological aspects perhaps in the test of Sprint 7, the getter and setter methods of Editor Component were tested, along with the key event and popup management, to check if they worked correctly. In Sprint number 8 its adaption to the PABRE-Man tool is tested and Editor Component is extended in this perspective to adapt itself correctly to the PABRE-Man tool. Technologically, in Sprint number 8, the component retrieval and adding to PABRE-Man tool's different Presentation Tier components, as well as the Editor Component's behavior, is tested. This test includes for instance, that the Editor Component corresponding to the JTextField which showed error message and set constant text when text attribute is void, behaves exactly the same. Once this functionality is checked to be ok, then the next functionality is tested until the whole Editor Component functionalities are proved.

A Test Set plan has to be made to check all functionalities of all artifacts added by this project to the PABRE-Man tool. Usually Test Set plans would start with an initial state picture of the tool before any type of work is done to the project as a reference to compare to as the development undergoes. However, being this project an extension of the tool by incorporation of new software and functionalities only related to few and specific union points with the actual tool, a reference snapshot of the tool does not acquire the complete relevance, and only is a reference of how those specific functionalities worked. In other words, as the PABRE tool did not have a glossary extension, no functionality comparison can be made, only the application of the extension can be tested in the points of the PABRE tool were it is applied and compared to the same points without this project's extension.

Task plan breakdown can be used to determine the test set plan followed during the development of this project. Testing shall be located temporarily at the end of every sprint were code has been developed. In difference with the task breakdown, instead of length and type of the specific tasks in every sprint, testing breakdown includes two different variables: if the task has been tested in a visual perspective or a data perspective. Visual perspective deals with look and feel and layout management or any specific Presentation Tier functionalities. Data perspective deals with data that is managed or introduced in the Business Tier or the Persistence Tier. Sometimes either visual or data perspective is senseless to be tested because it does not proceed for the task to be tested in such perspective and so it shall be stated, and sometimes the test will include both perspectives.

9.1 Test Set Task Breakdown

SPRINT NUMBER 3		
TASK	VISUAL	DATA
Business Tier testing	-	OK
Business Tier controller coding	-	OK

Notice how testing starts in Sprint number 3, where first artifacts are obtained.

SPRINT NUMBER 4		
TASK	VISUAL	DATA
Functional Requirement testing	-	OK
Business Tier integration	-	OK

Tests continue to deal with data perspective because no glossary Presentation Tier artifacts have been coded yet.

SPRINT NUMBER 5		
TASK	VISUAL	DATA
Hibernate persistence coding	-	OK
Functionality testing with persistence	-	OK
Persistence Load Test	-	OK

Persistence is tested fully in this Sprint.

SPRINT NUMBER 6		
TASK	VISUAL	DATA
Tree render of data	OK	-
Event management adding/editing/deleting	OK	OK

First visual testing to check data is rendered as required and data is managed according to visual events.

SPRINT NUMBER 7		
TASK	VISUAL	DATA
GlossaryInternalFrame creation	OK	-
Event management adding/editing/deleting	OK	OK
AutoHyponymChecker	OK	-
Editor Component Creation	OK	OK
Custom Dialog creation	OK	OK

Due to the fact Business Tier controller is coded and tested this is the first functional testing the glossary extension enhances. Test of independent components is taken here.

SPRINT NUMBER 8		
TASK	VISUAL	DATA
Tuning up functionalities & controllers	OK	OK
Possibility to evict glossary + hibernate	OK	OK
Tuning up Hibernate error bugs	-	OK
Editor Component Adaptation and extension	OK	-

In this sprint minor functionality bugs in controllers were tested and resolved. In a data perspective the possibility of discharge changes in the implicit glossary object was tested and proven in all three Tiers of the application – in the frame, indicating to evict the glossary in the dialog question, in the Business and data Tiers running eviction methods in Hibernate session and testing it with SQL Squirrel client- with satisfactory results. Editor component is tested visually searching for the getter and setter functionalities to work as previous components worked.

SPRINT NUMBER 9		
TASK	VISUAL	DATA
AutoHyponymChecker controller	OK	-
Extending Editor Component Substitution Event	OK	OK
Insert Editor Component to Pattern and Metric forms	OK	OK
Insert Editor Component to dialogs	OK	OK
adapt customized sizes of editor Components	OK	-

AutoHyponymChecker together with Editor Component is tested to evaluate if substitution is achieved in a visual perspective, substituting the invalid words in the Component's text attribute using AutoHyponymChecker's tools. Afterwards Editor Components were tested to be able to be added and adapted to the functionalities of the different PABRE-Man Presentation Tier elements.

SPRINT NUMBER 10		
TASK	VISUAL	DATA
Glossary XML exportation/importation	OK	OK
Adapt Toolbar and Internal frame position management	OK	-
Correct latter bugs	OK	OK

This is the test run to be able to import and export the glossary in XML format. Loading an invalid XML format file is required to check the error message is shown properly.

The tools toolbar is tested to open and close the different internal frames with special attention to the GlossaryInternalFrame. A simple management of open internal frames is also implemented and tested so that no horizontal conflicts – overlapping internal frames- arise when more than two different internal frames are open.

Visual and Data last bugs were also tested to be corrected in both perspectives resulting in complete correctness of the whole glossary extension.

10 Future Work

The Glossary extension for the PABRE-Man tool can be extended in the future. Due to the Three Tier Architecture Pattern, adding up or replacing code is much easier because functionalities are clearly segmented into artifacts and class objects.

For instance, if future work required an additional Editor Component, it just shall be instantiated as an attribute of the class - which behaves as a factory- and make sure the new component supports getters and setters of its own text attribute, as well as listeners to its Java text component, which will be also retrieved by the Editor Component class when required.

Other possible future extension of the Glossary subsystem includes a smart Domain Controller which will be consulted by the Glossary Internal Frame controller every time the Internal Frame is opened and also when it is closed. If the Glossary domain object notifies the Internal Frame controller the domain object has not been changed since the last persisting process the Internal Frame controller does not ask the user if he/she wants to save the glossary changes, since there are none. This is really practical for the user when he/she only wants to only consult the glossary seeking perhaps for a certain hypernym or hyponym, avoiding to tediously answering NO to the persistence dialog question. If changes are made to the glossary, obviously the question shall remain so the user can decide whether or not to save the changes to the glossary.

Finally other practical expansion of the glossary extension could be towards the Web Services ground. If for any reason the content of the glossary is to be required by any Web Service or framework – such as Google Web Toolkit-, it would be reasonable to pass the data through a JSON type exchange. For instance, defining the JSON schema as the XSD example the process of converting the data from the glossary to this schema is the same as for writing the XML file when the glossary is exported as a XML file. In pseudo code the algorithm shall be similar to the following:

```
Glossary: Map<String, String>;
Hypernyms: Multimap <String, String>
Description: Map <String, String>
String result=json_encode("GLOSSARY");

For each(element from Description){
 result:=result.append(json_encode("HYPERNYM"));
 result=result.append(element.hypernym);
 result:=result.append(json_encode("DESCRIPTION"));
 result:=result.append(json_end_close(element.description));

 Lhypo:=Hypernyms.get(element.hypernym))
       For each(hyponym from Lhypo) {
              result:=result.append(json_encode("HYPONYM"));
              result:=result.append(json_end_close(hyponym));
       }
   result:=result.append(json_end_close()); //close hypernym
}
result:=result.append(json_end_close());
```

Where uppercase words represent JSON constants and the auxiliary JSON encoding functions depend on the defined schema.

11 Conclusions

Addressing this project was an inspiring and thrilling challenge for me. It combines different aspects of the subjects learned through the Computer Science Degree from which this project is about. Software Engineering influence is predominant and ironically redundant realizing the project deals with a tool for managing Software Engineering artifacts and processes. Computing Theory is also present in the development of this project, whilst defining the domain of the Glossary model input data. Algorithmic analysis has been used to define and choose optimal data structures which increase software's efficiency thoroughly and have provided, in second instance, an easier development. Compilers has proven to be essential to the AutoHyponymChecker part, since its analysis is closely similar and linked to the way a compiler analyzes programs, tokenizing the input and parsing it as an Abstract Syntax Tree and processing it using the Visitor Pattern. Design of Web-based Systems has also been an important subject because the Scrum development methodology is basically taught in this subject with a practical focus. Database subject has also been relevant to understand the impact of the chosen design in the Data Tier and how to design this bottom level Tier. Planning and Management of Information Technology Projects and Systems subject has also influenced the course of this project as an emphasis on the changeability of project's requirements and how to manage that risk. Definitively many have been the subjects that have been aiding tools to manage all situations during this project. Similarly lots of work relating to other subjects has been done but has not reached to be useful to the project. Anyway, the degree's overall knowledge has been determinant to arrive to a project's satisfactory conclusion.

Additionally there have been lots of technological aspects I did not know about, and others I had heard of but really did not master them. Hibernate, SAX parsing, JDOM, and SQuirreL SQL client where technologies and libraries which I did not know much about before the beginning of this project. Java User Interface programming had been lightly touched during the degree and no other need had forced me to learn about it. It is true that in the degree I have learned about other User Interface frameworks or libraries such as Qt or OpenGL but had no knowledge intersection with Swing Java library. It has been thrilling for me to learn all these new skills and put them in practice as it is certainly inspiring to foresee a future of everlasting skill learning.

A special gratitude message is to be sent to the whole open source community which develop helpful and didactic software published its code under the GNU General Public License (GPL) [special thanks to the JOrtho [13] development team], as well as the many problem-solving contributions of specialized internet forums which definitely clarify out doubts and issues.

It has been great to contribute and realize this final degree project as a part of a real and currently used tool. The last words are to express my deep desire to guarantee the upmost success of the project hoping the tool to be used broadly.

12 Acknowledgments and thanks

I would like to use this space to, first of all, thank Carme Quer and Crisina Palomares for giving me the chance to develop this Final Degree Project under their supervision, guidance and support while developing this extension for the PABRE tool which they maintain. It has been a great pleasure to work shoulder to shoulder developing my contribution to the PABRE tool.

I want to also mention the deep effort of all the professors-from whom Carme Quer is one of them- which have spent their time and dedication by sharing their knowledge and enthusiasm with their students. Thanks to that, problem solving becomes easier and future becomes inspiring.

I would also like to thank and have a special mention all the different people I have met during the Degree and that have contributed with something positive to my person and skills either sharing either laboratory in some subject, project subjects or even enjoying the amenities between lectures.

On other perspective I would also like to thank and acknowledge to my mother and father for giving me the chance to gather from knowledge in every aspect of life, from the essentials to the most detailed topics, and for making it possible to have a wide and extensive opportunity to do so. This project is dedicated to you.

Finally, last but not least, quite the opposite, I shall want to give a special mention to a Person which is intended to be forgotten but nevertheless is and will be present in all our lives. May all the good things I do serve to praise Jehovah. (Psalms 83:18).

To all of you, thank you for your support without which this project could not have been done.

13 Bibliography

Note: All on-line resources are checked to be operational at least till today, January 16, 2014

[1] *PABRE-Man: Management of a Requirement Patterns Catalogue*, Cristina Palomares, Carme Quer, Xavier Franch. Software Engineering for Information Systems research group (GESSI) Universitat Politècnica de Catalunya (UPC)

[2]The PABRE tool http://www.upc.edu/gessi/PABRE/

[3] Hypernymy and Hyponym http://en.wikipedia.org/wiki/Hyponymy_and_hypernymy

[4] Java programming language http://www.java.com/en/download/whatis_java.jsp

[5] Hibernate http://hibernate.org/orm/

[6] Guava Library, Multimap

http://code.google.com/p/guava-libraries/wiki/CollectionUtilitiesExplained#Multimaps

[7] SAX http://docs.oracle.com/javase/tutorial/jaxp/sax/parsing.html

[8] Eclipse project http://www.eclipse.org/org/

[9] SQuirreL SQL client http://squirrel-sql.sourceforge.net/#overview

[10] Generating XML files using SAX and XSLT
http://www.developer.com//xml/article.php/2108031/Transforming-Flat-Files-To-XML-With-SAX-and-XSLT.htm

[11] Scrum Methodology https://www.scrum.org/Resources/What-is-Scrum

[12] Add Spellchecker to Java applications http://www.wintertree-software.com/spell-check/java/

[13] JOrtho - a Java spell-checking library http://jortho.sourceforge.net/

[XML] *Manual Imprescindible XML Edición 2012* Miguel Ángel Acera García , Ediciones Anaya Multimedia , 2011, ISBN 9788441529601

[HIB1] *Harnessing hibernate* -James Elliott, Ryan Fowler, and Tim O'Brien, - O'Reilly, 2008, ISBN: 9780596517724

[HIB2] *Hibernate recipes: a problem-solution approach*- Srinivas Guruzu, Gary Mak,- Apress, 2010 – ISBN: 9781430227960

[HIB3] *Beginning Hibernate* - Jeff Linwood and Dave Minter- Apress, 2010, ISBN: 9781430228509

[GoF] *Design Patterns: Elements of Reusable Object-Oriented* Software -Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides - Addison Wesley, 1995, ISBN: 0201633612

[DSS] *Diseño de sistemas software en UML*- Cristina Gómez, Enric Mayol, Antoni Olivé, Ernest Teniente –Edicions UPC, 2003, ISBN: 8483017245

[Lar03] *UML Y Patrones* -Craig Larman- Prentice Hall, 2003, ISBN: 8420534382

[SWN] *The JFC swing tutorial: a guide to constructing GUIs* / Kathy Walrath , Addison-Wesley, 2004, ISBN: 0201914670