LeXFlow: a System for Cross-fertilization of Computational Lexicons

Maurizio Tesconi and **Andrea Marchetti**
CNR-IIT
Via Moruzzi 1, 56024 Pisa, Italy
{maurizio.tesconi,andrea.marchetti}@iit.cnr.it

Francesca Bertagna and **Monica Monachini** and **Claudia Soria** and **Nicoletta Calzolari**
CNR-ILC
Via Moruzzi 1, 56024 Pisa, Italy
{francesca.bertagna,monica.monachini,
claudia.soria,nicoletta.calzolari}@ilc.cnr.it

Abstract

This demo presents LeXFlow, a workflow management system for cross-fertilization of computational lexicons. Borrowing from techniques used in the domain of document workflows, we model the activity of lexicon management as a set of workflow types, where lexical entries move across agents in the process of being dynamically updated. A prototype of LeXFlow has been implemented with extensive use of XML technologies (XSLT, XPath, XForms, SVG) and open-source tools (Cocoon, Tomcat, MySQL). LeXFlow is a web-based application that enables the cooperative and distributed management of computational lexicons.

1 Introduction

LeXFlow is a workflow management system aimed at enabling the semi-automatic management of computational lexicons. By management we mean not only creation, population and validation of lexical entries but also integration and *enrichment* of different lexicons.

A lexicon can be enriched by resorting to automatically acquired information, for instance by means of an application extracting information from corpora. But a lexicon can be enriched also by resorting to the information available in another lexicon, which can happen to encode different types of information, or at different levels of granularity. LeXFlow intends to address the request by the computational lexicon community for a change in perspective on computational lexicons: from static resources towards *dynamically configurable multi-source entities*, where the content of lexical entries is dynamically modified and updated on the basis of the integration of knowledge coming from different sources (indifferently represented by human actors, other lexical resources, or applications for the automatic extraction of lexical information from texts).

This scenario has at least two strictly related prerequisites: i) existing lexicons have to be available in or be mappable to a standard form enabling the overcoming of their respective differences and idiosyncrasies, thus making their mutual comprehensibility a reality; ii) an architectural framework should be used for the effective and practical management of lexicons, by providing the communicative channel through which lexicons can really communicate and share the information encoded therein.

For the first point, standardization issues obviously play the central role. Important and extensive efforts have been and are being made towards the extension and integration of existing and emerging open lexical and terminological standards and best practices, such as EAGLES, ISLE, TEI, OLIF, Martif (ISO 12200), Data Categories (ISO 12620), ISO/TC37/SC4, and LIRICS. An important achievement in this respect is the MILE, a meta-entry for the encoding of multilingual lexical information (Calzolari et al., 2003); in our approach we have embraced the MILE model.

As far as the second point is concerned, some initial steps have been made to realize frameworks enabling inter-lexica access, search, integration and operability. Nevertheless, the general impression is that little has been made towards the development of new methods and techniques for the concrete interoperability among lexical and textual resources. The intent of LeXFlow is to fill in this gap.

2 LeXFlow Design and Application

LeXFlow is conceived as a metaphoric extension and adaptation to computational lexicons of XFlow, a framework for the management of document workflows (DW, Marchetti et al., 2005).

A DW can be seen as a process of *cooperative authoring* where the document can be the goal of the process or just a side effect of the cooperation. Through a DW, a document life-cycle is tracked and supervised, continually providing control over the actions leading to document compilation In this environment a document travels among *agents* who essentially carry out the pipeline receive-process-send activity.

Each lexical entry can be modelled as a document instance (formally represented as an XML representation of the MILE lexical entry), whose behaviour can be formally specified by means of a document workflow type (DWT) where different agents, with clear-cut roles and responsibilities, act over different portions of the same entry by performing different tasks.

Two types of agents are envisaged: *external agents* are human or software actors which perform activities dependent from the particular DWT, and *internal agents* are software actors providing general-purpose activities useful for any DWT and, for this reason, implemented directly into the system. Internal agents perform general functionalities such as creating/converting a document belonging to a particular DWT, populating it with some initial data, duplicating a document to be sent to multiple agents, splitting a document and sending portions of information to different agents, merging duplicated documents coming from multiple agents, aggregating fragments, and finally terminating operations over the document. An external agent *executes* some processing using the document content and possibly other data, e.g. *updates* the document inserting the results of the preceding processing, *signs* the updating and finally *sends* the document to the next agent(s).

The state diagram in Figure 1 describes the different states of the document instances. At the starting point of the document life cycle there is a creation phase, in which the system raises a new instance of a document with information attached.

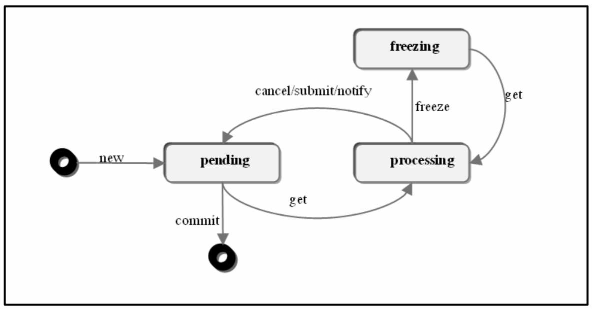


Figure 1. Document State Diagram.

The document instance goes into *pending* state. When an agent gets the document, it goes into *processing* state in which the agent compiles the parts under his/her responsibility. If the agent, for some reason, doesn't complete the instance elaboration, he can save the work performed until that moment and the document instance goes into *freezing* state. If the elaboration is completed (submitted), or cancelled, the instance goes back into *pending* state, waiting for a new elaboration.

Borrowing from techniques used in DWs, we have modelled the activity of lexicon management as a set of DWT, where lexical entries move across agents and become dynamically updated.

3 Lexical Workflow General Architecture

As already written, LeXFlow is based on XFlow which is composed of three parts: i) the Agent Environment, i.e. the agents participating to all DWs; ii) the Data, i.e. the DW descriptions plus the documents created by the DW and iii) the Engine. Figure 2 illustrates the architecture of the framework.

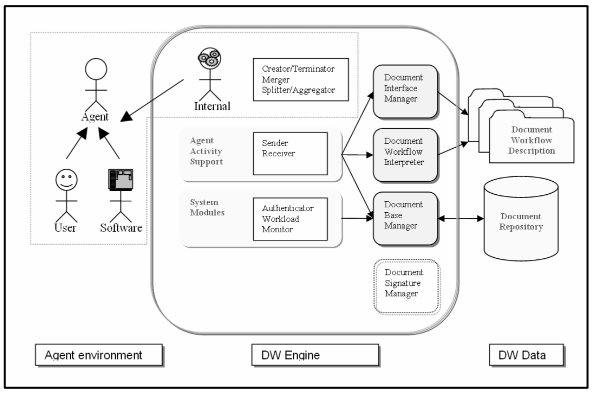


Figure 2. General Architecture.

The DW environment is the set of human and software agents participating to at least one DW.

The description of a DW can be seen as an extension of the XML document class. A class of documents, created in a DW, shares the schema of their structure, as well as the definition of the procedural rules driving the DWT and the list of the agents attending to it. Therefore, in order to describe a DWT, we need four components:

- a schema of the documents involved in the DWT;
- the agent roles chart, i.e. the set of the external and internal agents, operating on the document flow. Inside the role chart these agents are organized in roles and groups in order to define who has access to the document. This component constitutes the DW environment;
- a document interface description used by external agents to access the documents. This component also allows checking access permissions to the document;
- a document workflow description defining all the paths that a document can follow in its life-cycle, the activities and policies for each role.

The document workflow engine constitutes the run-time support for the DW, it implements the *internal agents,* the support for *agents' activities*, and some *system modules* that the external agents have to use to interact with the DW system. Also, the engine is responsible for two kinds of documents useful for each document flow: the *documents system logs* and the *documents system metadata.*

4 The lexicon Augmentation Workflow Type

In this section we present a first DWT, called "lexicon augmentation", for dynamic augmentation of semantic MILE-compliant lexicons. This DWT corresponds to the scenario where an entry of a lexicon *A* becomes enriched via basically two steps. First, by virtue of being mapped onto a corresponding entry belonging to a lexicon *B*, the entry$_{(A)}$ inherits the semantic relations available in the mapped entry$_{(B)}$. Second, by resorting to an automatic application that acquires information about semantic relations from corpora, the acquired relations are integrated into the entry and proposed to the human encoder.

In order to test the system we considered the Simple/Clips (Ruimy et al., 2003) and ItalWordNet (Roventini et al., 2003) lexicons.

An overall picture of the flow is shown in Figure 3, illustrating the different agents participating to the flow. Rectangles represent human actors over the entries, while the other figures symbolize software agents: ovals are internal agents and octagons external ones. The functionality offered to human agents are: display of MILE-encoded lexical entries, selection of lexical entries, mapping between lexical entries be-

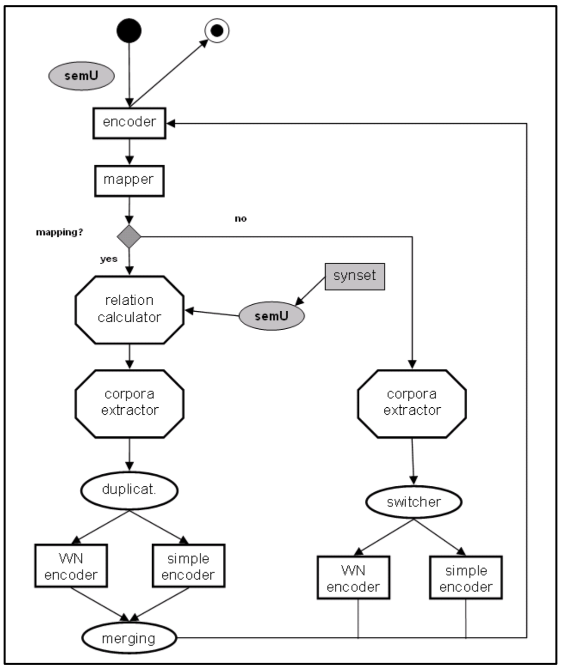


Figure 3. Lexicon Augmentation Workflow.

longing to different lexicons[1], automatic calculations of new semantic relations (either automatically derived from corpora and mutually inferred from the mapping) and manual verification of the newly proposed semantic relations.

5 Implementation Overview

Our system is currently implemented as a web-based application where the human external agents interact with system through a web browser. All the human external agents attending the different document workflows are the users of system. Once authenticated through username and password the user accesses his workload area where the system lists all his pending documents (i.e. entries) sorted by type of flow.

The system shows only the flows to which the user has access. From the workload area the user

[1] We hypothesize a human agent, but the same role could be performed by a software agent. To this end, we are investigating the possibility of automatically exploiting the procedure described in (Ruimy and Roventini, 2005).

can browse his documents and select some operations

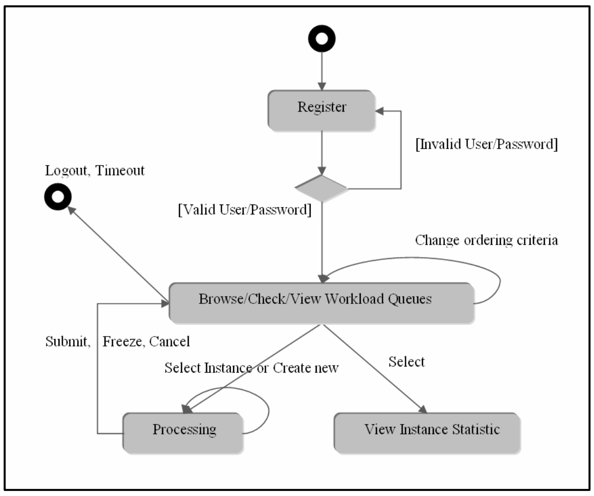


Figure 4. LeXFlow User Activity State Diagram.

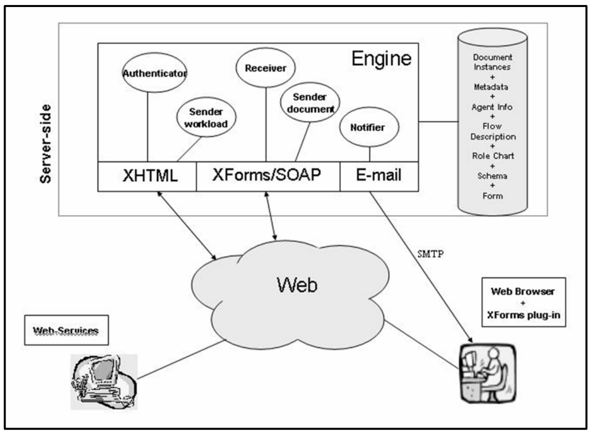
such as: selecting and processing pending document; creating a new document; displaying a graph representing a DW of a previously created document; highlighting the current position of the document. This information is rendered as an SVG (Scalable Vector Graphics) image. Figure 5 illustrates the overall implementation of the system.

5.1 The Client Side: External Agent Interaction

The form used to process the documents is rendered with XForms. Using XForms, a browser can communicate with the server through XML documents and is capable of displaying the document with a user interface that can be defined for each type of document. A browser with XForms capabilities will receive an XML document that will be displayed according to the specified template, then it will let the user edit the document and finally it will send the modified document to the server.

5.2 The Server Side

The server-side is implemented with Apache Tomcat, Apache Cocoon and MySQL. Tomcat is used as the web server, authentication module (when the communication between the server and the client needs to be encrypted) and servlet container. Cocoon is a publishing framework that uses the power of XML. The entire functioning of Cocoon is based on one key concept: component pipelines. The pipeline connotes a series of events, which consists of taking a request as input, processing and transforming it, and then giving the desired response. MySQL is used for storing and retrieving the documents and the status of the documents.



Server-side
Engine
Authenticator
Receiver
Sender workload
Sender document
Notifier
XHTML
XForms/SOAP
E-mail
Document Instances + Metadata + Agent Info + Flow Description + Role Chart + Schema + Form
SMTP
Web
Web-Services
Web Browser + XForms plug-in

Figure 5. Overall System Implementation.

Each software agent is implemented as a web-service and the WSDL language is used to define its interface.

References

Nicoletta Calzolari, Francesca Bertagna, Alessandro Lenci and Monica Monachini, editors. 2003. *Standards and Best Practice for Multilingual Computational Lexicons. MILE (the Multilingual ISLE Lexical Entry).* ISLE Deliverable D2.2 & 3.2. Pisa.

Andrea Marchetti, Maurizio Tesconi, and Salvatore Minutoli. 2005. XFlow: An XML-Based Document-Centric Workflow. In *Proceedings of WI-SE'05*, pages 290- 303, New York, NY, USA.

Adriana Roventini, Antonietta Alonge, Francesca Bertagna, Nicoletta Calzolari, Christian Girardi, Bernardo Magnini, Rita Marinelli, and Antonio Zampolli. 2003. ItalWordNet: Building a Large Semantic Database for the Automatic Treatment of Italian. In Antonio Zampolli, Nicoletta Calzolari, and Laura Cignoni, editors, *Computational Linguistics in Pisa*, Istituto Editoriale e Poligrafico Internazionale, Pisa-Roma, pages 745-791.

Nilda Ruimy, Monica Monachini, Elisabetta Gola, Nicoletta Calzolari, Cristina Del Fiorentino, Marisa Ulivieri, and Sergio Rossi. 2003. A Computational Semantic Lexicon of Italian: SIMPLE. In Antonio Zampolli, Nicoletta Calzolari, and Laura Cignoni, editors, *Computational Linguistics in Pisa*, Istituto Editoriale e Poligrafico Internazionale, Pisa-Roma, pages 821-864.

Nilda Ruimy and Adriana Roventini. 2005. Towards the linking of two electronic lexical databases of Italian. In *Proceedings of L&T'05 - Language Technologies as a Challenge for Computer Science and Linguistics*, pages 230-234, Poznan, Poland.