TALN 2006, Leuven, 10-13 avril 2006

Computer Tools for the Management of Lexicon-Grammar Databases

Javier M. Sastre Martínez

IGM – Université de Marne-la-Vallée, F 77454 Marne-la-Vallée Cedex 2
sastre@univ-mlv.fr

Résumé Le lexique grammaire est une méthode systématique d'analyse et de représentation des structures de phrase élémentaire d'une langue naturelle ; son produit: des grandes collections de dictionnaires syntaxiques électroniques ou tables de lexique-grammaire (LGTs). Du travail collaboratif à très long terme est nécessaire pour achever la description d'une langue. Cependant, les outils informatiques de gestion de LGTs actuels ne remplissent les besoins suivants : intégration automatique de données multisource, control de cohérence de données et de versions, filtrage et tri, formats d'échange, gestion couplée des données et de la documentation, interfaces graphiques (GUIs) dédiées et gestion d'utilisateurs et contrôle d'accès. Dans cet article nous proposons une solution basée sur PostgreSQL et/ou MySQL (systèmes de gestion de bases de données libres), Swing (une librairie pour la programmation de GUIs en Java), JDBC (API pour la connectivité de Java aux bases de données), et StAX (API pour l'analyse et la création des documents en XML).

Abstract Lexicon grammar is a systematic method for the analysis and the representation of the elementary sentence structures of a natural language; its product: large collections of syntactic electronic dictionaries or lexicon-grammar tables (LGTs). In order to describe a language, very long term collaborative work is required. However, the current computer tools for the management of LGTs do not fulfill the following requirements: automatic integration of multisource data, data coherence and version control, filtering and sorting, exchange formats, coupled management of data and documentation, dedicated graphical interfaces (GUIs) and user management and access control. In this paper we propose a solution based on PostgreSQL and/or MySQL (open source database management systems), Swing (a GUI toolkit for Java), JDBC (the API for Java database connectivity) and StAX (an API for the analysis and generation of XML documents).

Mots-clés : Table de lexique-grammaire, base de données, interface graphique, XML
Keywords: Lexicon-grammar table, database, graphical interface, XML

1 Introduction

Lexicon grammar is a model of syntax focused on the elementary sentences of a natural language (Gross, 1996). Its major principle is that the unit of meaning is the sentence, but the word or the words composing the predicative element (PE) of a sentence constraint its structure. Natural language grammars are too irregular to be described by a set of general rules (Gross M. 1997). Rather than that, the lexicon grammar approach consists in enumerating the sentence structures for each use of a sentence PE. In order to compact and split the list into components easy to handle, the structures are classified by similarity into lexicon-grammar tables (Leclère, 2002) or LGTs. Each table is associated to a common sentence structure; table columns are associated to variants or differential properties of the common structure; table rows illustrate the possible structures of a specific sentence PE. Two kinds of columns are used: boolean and text. The former indicate the acceptability or unacceptability of a structure property; e.g.: *N1 =: Qu Psubj* meaning that the direct object may be a clause in subjunctive mood introduced by the conjunction *que* (e.g.: *Max aime que Marie lui fasse des câlins* ≈ Max likes that Marie gives him cuddles[1]). The latter, to point out the use of a specific word within the sentence (e.g.: the preposition introducing a verb argument; in *tomber amoureux de quelqu'un*, only the preposition *de* can introduce the object complement *quelqu'un* as well as in the English version *to fall in love with somebody* only the preposition *with* can introduce the object complement *somebody*). LGTs can be automatically transformed into finite state automata (FSTs) in order to perform parsing (Roche, 1993, Constant 2003). Each table is associated to a set of parameterized FSTs[2], which recognize every variant of the common sentence structure. By instantiating the parameters by the values contained in each row we obtain the set of FSTs recognizing each particular structure. Other ongoing projects intend to use LGTs with other parsing frameworks (Garden et al., 2005).

Summarizing, the representation of syntactic structures by means of LGTs benefits from three important properties:

1. Simplicity: each cell contains a single unit of information, namely the acceptability or unacceptability of a property or the potential use of a word, and information units are grouped into tables which can be easily viewed or printed.

2. Exhaustiveness: a large coverage of a natural language can be achieved by the incremental construction of LGTs, yet many irregularity instances can be taken into account.

3. Potential exploitability for automatic parsing: it is possible to automatically transform the tables into FSTs for corpus parsing given a set of parameterized FSTs.

---

[1] Literal translations.

[2] Parameterized FSTs are built manually and tested against corpus.

2 State of the art in computer tools for the management of lexicon grammar databases

LGTs are presently created and maintained using common spreadsheet software, mainly Microsoft (MS) Excel but also OpenOffice. A web interface is used for public table view which implements a rudimentary LGT database indexed by sentence PEs and table codes. The employed methodology presents several limitations mainly derived from the use of such tools:

1. Property values are hard-coded as text sequences: '+' and '-' symbols represent acceptability and unacceptability values respectively (Figure 1); unset values (nulls) are represented by '~' symbols. Editor applications are not explicitly aware of the data characteristics, thus do not offer a dedicated management depending on data types (e.g.: appropriate representation of true/false values, correctness control of value codes, etc).

2. These tools do not support network shared objects; every LGT is locally developed and registered as a file in the author's computer. In order to share a LGT through the network the correspondent file is copied into a public web folder, where its access is limited to read-only by means of a web interface (Figure 2). The integration of different data sources must be manually done by locating the last version files, mixing the modifications done by different authors and recollecting them within a public web folder. As the number of files, different file versions and different file copies increases, the integration of the global database is complicated.

3. LGTs can reach a size of a thousand rows and forty columns, resulting in forty-thousand information units within a single table. Under these conditions, filtering of columns and rows becomes a need rather than an asset. Selective data access is limited to the search within a single table of text sequences potentially contained in a single cell. The web interface allows direct access to the table or tables containing a specified PE.

4. Problems of software and data portability: MS Excel is meant to be run only on MS Windows's platforms[3] and uses, by default, a proprietary file format[4]. MS. Excel allows for exporting into Unicode (Allen, 2003) plain text files with some extra manual effort, but a more appropriate solution would be the default use of an XML-based exchange format (Sastre, 2005).

5. Just for distributional verbs there are already sixty LGTs available, which jointly contain a set of 428 different properties. Each table and each property is represented

---

[3] MS. Windows's applications may work on other platforms thanks to Windows emulators or Windows software compatibility layers (e.g.: Wine); however such scenarios may be unacceptable due to the potential emulation errors and the added complexity to software utilisation.

[4] There exist tools able to load MS. Excel files (e.g.: OpenOffice), but errors due to format misinterpretation may happen. Those tools are usually not able to modify MS. Excel files, thus access is restricted to read-only.

by a short code, which tries to reflect the nature of the referenced object. However, codes may not always be easily interpreted without the aid of documentation; for example, the code *Nnr* (non-restricted noun) indicates if the sentence may accept any subject independently of its semantic nature (e.g: human or not human) or syntactic structure, namely a completive or an infinitival clause (Gross, 1975). Direct access to documentation fragments would be useful not only for the interpretation of existent data but also to facilitate the creation of new documentation. Current tools provide direct access to documentation about tables but not about finer-grained objects like columns. Row documentation is limited to an extra field within the table containing a general sentence example (Figure 1). Examples for each cell could be added by increasing the number of example columns, but the resultant tables would be too big to be easily handled without selective data access support.

6. There is no inherent data access control or user management. Private files are simply not distributed. If a file or a data subset of a file is to be shared, a new file containing the required data is created and placed within a public web folder. This complicates data maintenance because of the increasing number of copies to keep up to date.

N0 =: Nhum	N0 =: Nnc	<ENT>	aux =: avoir	aux =: être	1	N1 =: Qu Psubj	Tc =: passé	Vc =: devoir	V0-inf W = Ppv	<OPT>Exemple
+	-	achever	+	-	de	-	-	-	-	Max achève de peindre le mur
+	+	aller	-	-		-	-	+	-	Max va partir
+	-	aller	-	+	jusqu'à	-	-	-	-	La pluie va jusqu'à tomber
+	+	ne aller Nég	-	+	sans	+	-	+	-	Cette mesure n'ira pas sans créer des troubles
~	~	s'apprêter	~	~	à	~	~	~	~	La pluie s'apprête à tomber
+	+	arrêter	+	-	de	-	-	-	-	Luc arrête de travailler

Figure 1: Classic representation of a LGT fragment (LGT 1) under MS. Excel

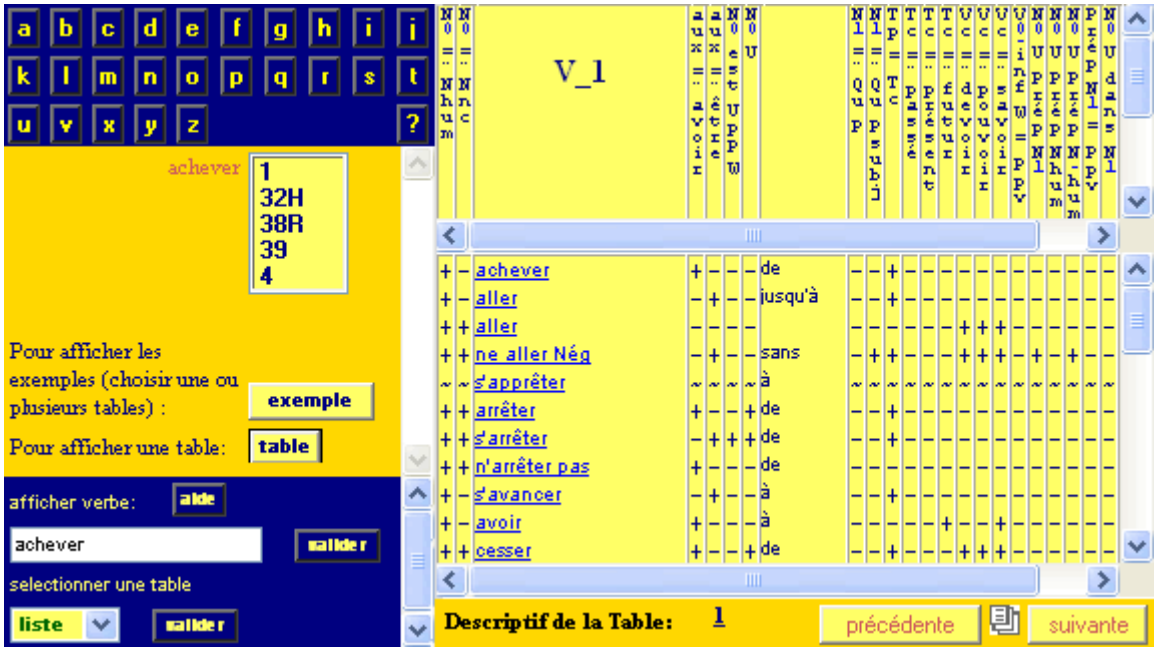

Figure 2: Web interface for the visualisation of LGTs

7. The conception of a new LGT involves the identification of a sentence structure class and its possible variants which are represented by property codes. Some properties are extensively reused whilst others need to be created for dealing with new scenarios. Due to the absence of a data coherence control and the increasing number of properties, it happens that different codes are used to represent the same property in different tables. This may not be problem for the human reader, but it is a serious obstacle to automatic management.

3 Computer tools proposal

We propose a computer tool solution based in a Java web/local client interface from which the user will be able to manage shared lexicon-grammar databases contained in a PostgreSQL or a MySQL database server. Clients use JDBC to communicate with the database server. Each part is explained in detail in the following subsections.

3.1 Java web/local applications

A very convenient feature of Java (Eckel, 2002) is the possibility of embedding Java applications within web pages (Java applets). Applets can be downloaded from the web and directly executed on several platforms just by opening their embedding web pages. An important drawback is the potential danger of executing malicious code, since we may not have any control over the software origin. That is the reason why applets are executed, by default, in a restricted resource access environment called "sandbox" (Gong, 1998). However, it is possible to digitally sign our software packages in order to provide a mean of ensuring their authenticity. Once the software packages are signed and deployed on a public web server, it is up to the user to trust or not the software web distributor in order to grant the same resource access as local applications. By adding some little source code to our typical Java applications and signing the correspondent software packages it is possible to use the same software package for local use as well as for web direct execution. This permits the user to directly execute our application through the web, thus avoiding the software installation, or to download the package and execute it locally in order to avoid network transfer overhead. Both scenarios have been taken into account for our LGT computer tools, the former for facilitating the diffusion of LGTs and providing an easy and fast way to access them, the latter for users who constantly work with them. Instead of having to develop two separate tools, we have developed a single one able to deal with both scenarios.

3.2 The Swing graphical interface

Swing (Loy, 2002) provides a complete framework for the development of graphical interfaces in Java. Amongst others, it gives a good support for the implementation of tabular data graphical interfaces. One important requirement was to be able to horizontally scroll a set of columns whilst keeping static the other ones. Since the LGTs may be larger than the computer screen, scrolling is necessary for viewing all the data. However the column or columns containing the PE must remain in view so that we can know at anytime which PE each row corresponds to. The column containing the example sentence may be required to remain as well. In Figure 3 we can see a LGT split into four column groups: properties affecting the sentence subject, the PE, complement properties and the examples.

Another important requirement was to handle LGTs for any language. This is quite problematic due to the existence of different character coding tables for different language alphabets. The appropriate character coding table must be selected in order to correctly draw the correspondent characters, but it may be also necessary to mix different alphabets within the same table representation (e.g.: to compare tables of different languages). The unique homogeneous solution is the use of Unicode, a universal character coding system. Java natively supports Unicode and Java Swing can render Unicode symbols without further complication (Figure 4).

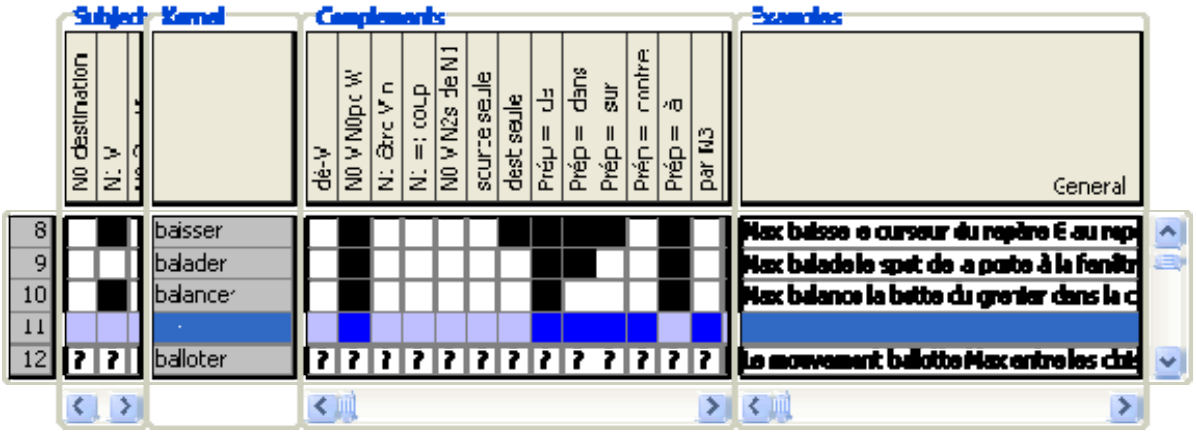

Figure 3 : Fragment of LGT 38L, highlighted line correspondent to verb *balancer*

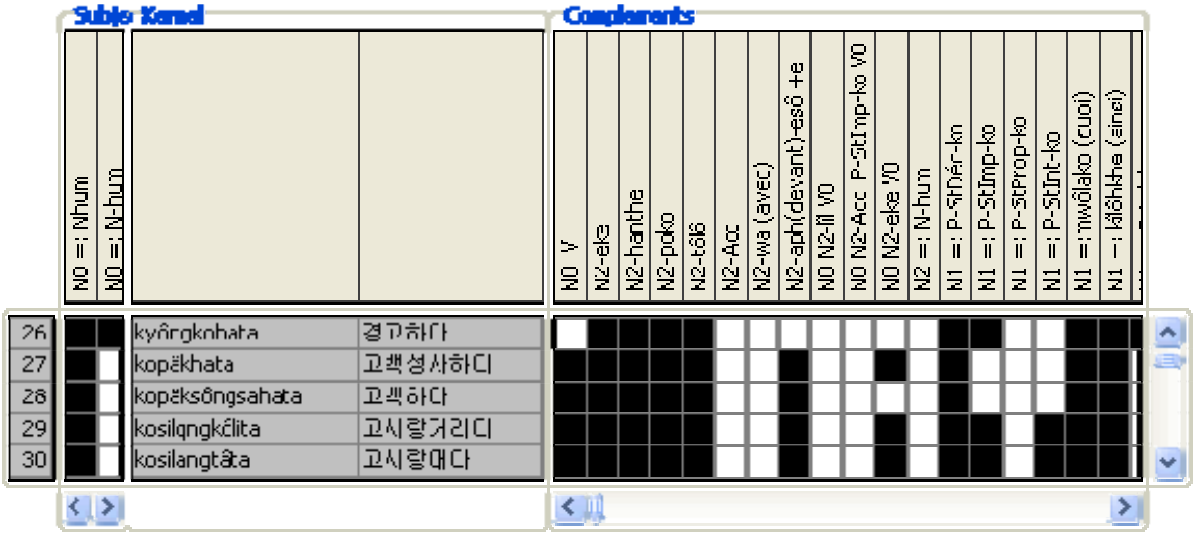

Figure 4 : Fragment of LGT for Korean verbs. A transcription to Latin characters is given.

The developed graphical interface represents boolean properties in a specific manner: black squares for acceptability, white squares for unacceptability and question marks for nulls. The proposed representation is more convenient for detecting value patterns (e.g.: in Figure 4 rows 29 and 30 have the same property values; several columns seem to have constant values). Invalid codes for boolean values cannot be introduced since the interface responds uniquely to ‘T’ (true), ‘F’ (false) or DEL (unknown value) key presses.

3.3 The lexicon-grammar database

LGTs can be easily inserted into database systems (Ullman, 1979) due to their similarity with respect to relational tables. Nowadays there exist reliable and fully functional open source implementations of database management systems (DBMS), mainly PostgreSQL (PostgreSQL, 2005) and MySQL (MySQL, 2005). These systems receive and process requests from client programs formulated in SQL (Structured Query Language) not only for data creation, access and modification but also for statistical analysis. An immediate application of LGT statistical analysis is the study of candidate LGT subclasses (decomposition of LGTs into sets of rows sharing similar value patterns).

Both PostgreSQL and MySQL do correctly manage concurrent database access thanks to the use of ACID transactions (Häder, 1983). Both provide a graphical interface for the management of generic databases, users, and access rights; LGTs are created within a database server from which network access is possible depending on the set permissions. Both provide data backup mechanisms. Both allow computer programs written in Java or C++, amongst other languages, direct database access for automatic data processing.

LGT properties are managed by maintaining within the database server a table of every existent property and its documentation. During the creation of new LGTs, current properties can be easily consulted in order to avoid the assignation of multiple codes to the same property. If a new property is to be created, it is added to the property table with its correspondent documentation. This mechanism provides a fast and direct access to property characteristics, both useful for the creation and the consultation of LGTs. A table containing the code and documentation of each LGT is also maintained.

3.4 Communicating with the database through JDBC

The JDBC API provides a universal mechanism to database access for Java clients (Fisher, 2002). The same Java program may access different database systems by loading the correspondent JDBC driver. Basically, a JDBC driver makes it possible to connect to a data source, send SQL statements and process the results. Since both PostgreSQL and MySQL provide such a driver, our LGT management system can rely on both systems without having to change a single line of source code. Since it is based on SQL, very few commands are needed in order to perform complex operations on LGTs. For making things even easier, it is even not necessary to have a perfect knowledge of SQL: the graphical interfaces included with PostgreSQL and MySQL show the SQL statement correspondent to each one of the requests we compose with mouse clicks and key presses.

3.5 XML data exchange interface

XML (Skonnard, 2001) constitutes a convenient way for the representation of data to be exchanged amongst computer applications. XML, Unicode and Java technologies converge into a set of multiple free tools for the generation, analysis and validation of XML documents, each one following a different approach:

1. Document Object Model (DOM) is a platform independent interface that exposes XML documents as a tree structure. It loads the whole XML document in memory

and then provides an easy way to access specific parts. Random access to different document parts is enabled with a memory cost proportional to the document size

2. XML Schema (Vlist, 2002) is a XML-based language for the definition of XML formats. JAXB (Amstrong, 2005) is a metacompiler that automatically generates Java code for the analysis, validation and generation of XML documents following a specific XML Schema description. This is a Java specific solution similar to DOM: a Java object tree is created in memory representing the whole document.

3. The XMLEncoder (Philip, 2005) provides a way to XML automatic serialization of Java Beans (DeSoto, 1997). Java Beans are Java classes that follow some naming conventions. Using the Java reflection mechanism (Green, 2000), XMLEncoder can find out which data must be registered for the object serialization by inspecting the object structure. XML documents generated this way are intended to be decoded by the XMLDecoder, which in turn recreates the original Java object. This is the easiest but most Java specific solution to XML serialization: we are not able to choose the XML format; it is automatically generated for the serialization and recreation of Java objects by Java applications.

4. Document streaming consist in serially parsing XML documents. Random access to specific parts is not allowed but the resource requirements are often smaller, mainly if we work with big data objects. Parsed elements may be *pushed* or *pulled* from the parser towards the application. In the first case, the application passively receives and deals with the events generated by the parser. In the second case, the application actively requests the XML parsers for data fragments. Examples of push and pull parsers for Java are, respectively, SAX (Brownell, 2002) and StAX (Amstrong, 2005). SAX provides document parsing only, StAX both parsing and generation.

Since we do not require random document access, StAX represents the most suitable solution because it is the minimal resource consumer, the source code required for pull parsing models is usually simpler than for push parsing models, it can be used not only for reading but also for writing XML documents and it is not restricted to the use of Java technologies (other computer applications developed with other computer languages, like C++, may interact with XML-serialized LGTs).

3.5.1 The XML-LGT format

LGTs are represented by <lgt> tags, e.g.: <lgt name="38L" version="1.0" docUri=" http://···/38L.html">. The attributes *name*, *version* and *docUri* indicate, respectively, the table name, format version and the URI where to locate the table documentation. <lgt> tags may contain a set of <column> tags, one per property column, followed by a set of <row> tags, one per table row. <column> tags describe the properties of a column by the following attribute set: *type* indicates the property type ("boolean" or "text"), *isPrimaryKey* indicates whether a column contains the row identifiers (the sentence PEs) and *acceptsNulls* whether a column may contain null values (usually non-primary key columns). <row> tags contain a set of <value> text tags, one per column. Empty content is used to represent null values (e.g.: <value/>). "true" and "false" strings indicate true and false values for boolean type columns (e.g.: <value>true</value>). For text type columns, the correspondent text is the content of the <value> tag (e.g.: <value>aimer</value>). This representation format is appropriate for loading tables within a database by a document streaming parser, since the information is

accessed in the optimal order required by a database to create new tables: table name, column properties and row values.

4 Conclusion

Lexicon grammar is a promising methodology for the collection of syntactic information which can be later used for automatic parsing. As the work in this field has advanced, the current used computer tools have begun to show their limitations. In order to ensure the progress of lexicon-grammar based research, we have proceeded to study the new requirements, to analyze the current existent free tools which could best fit them, and to develop a LGT management system based on them. The use of a centralized database serves as a nexus between linguistic data compilation from human experts and its exploitation by computer programs. Future work will consist in the study of optimal algorithms for the conversion of LGTs into FSTs and their implementation for lexicon-grammar based parsing.

Acknowledgements

We would like to thank Pr. E. Laporte for his continuous help. This work has been partially supported by CNRS, the French Ministry of Industry and the Institut de linguistique française.

References

ALLEN J., BECKER J. (2003). *The Unicode Standard Version 4.0*. The Unicode Consortium, http://www.unicode.org/versions/Unicode4.0.0/bookmarks.html

AMSTRONG E., BODOFF S., CARSON D. (2005). *The Java^{tm} Web Services Tutorial*. Sun Microsystems, http://java.sun.com/webservices/docs/1.6/tutorial/doc/JavaWSTutorial.pdf

BROWNELL D. (2002). *SAX2*. Sebastopol, CA, USA: O'Reilly & Associates, Inc., Sebastopol, ISBN 0596002378.

CONSTANT M. (2003). Converting Linguistic Systems of Relational Matrices into Finite State Transducers. *Proceedings of the EACL Workshop on Finite-State Methods in Natural Language Processing*, Budapest, 456-465.

DESOTO A. (1997). *Using the Beans Development Kit 1.0: A Tutorial*. Sun Microsystems, http://java.sun.com/products/javabeans/docs/Tutorial-Sep97.pdf

ECKEL B. (2002). *Thinking in Java*. Prentice-Hall, ISBN: 0131002872.

FISHER M. (2002). *JDBC^{TM} Database Access*. Sun Microsystems, http://java.sun.com/docs/books/tutorial/jdbc/index.html

GARDEN C., GUILLAUME B., PERRIER G., FALK I. (2005). Maurice Gross' Grammar Lexicon and Natural Language Processing. *Proceedings of the 2nd Language & Technology Conference*, Poznan, Poland

GONG L. (1998). *Java™ 2 Platform Security Architecture*. Sun Microsystems, http://java.sun.com/j2se/1.5.0/docs/guide/security/spec/security-spec.doc.html

GREEN D. (2000). *Reflection API Tutorial*. Sun Microsystems, http://java.sun.com/docs/books/tutorial/reflect/

GROSS M. (1975). *Méthodes en syntaxe : régime des constructions complétives*. Paris : Hermann, ISBN 270561365X.

GROSS M. (1996). Lexicon Grammar. *Concise Encyclopedia of Syntactic Theories*, K. Brown & J. Miller, Cambridge, Pergamon Press, 244-258.

GROSS M. (1997). The Construction of Local Grammars. *Finite State Language Processing*, Cambridge, Mass, The MIT Press, Roche E. and Schabes Y., 329-352.

HÄDER T., REUTER A. (1983). Principles of Transaction-Oriented Database Recovery. *ACM Computing Surveys 15:4*, ACM Press: New York, NY, USA, 287-317

LECLERE C. (2002). Organization of the Lexicon-Grammar of French verbs. *Lingvisticae Investigationes 25:1*, Amsterdam/Philadelphia : John Benjamins, 29-48.

LOY M., ECKSTEIN R., WOOD D. (2002). *Java Swing*. O'Reilly, ISBN: 0596004087.

MYSQL AB. (2005). *MySQL 5.1 Reference Manual*. MySQL AB., http://downloads.mysql.com/docs/refman-5.1-en.a4.pdf

PHILIP M. (2005). *Using XML Encoder*. Sun Microsystems, http://java.sun.com/products/jfc/tsc/articles/persistence4/

POSTGRESQL GDG. (2005). *PostgreSQL 8.1.0 Documentation*. The PostgreSQL Global Development Group, http://www.postgresql.org/files/documentation/pdf/8.1/postgresql-8.1-A4.pdf

ROCHE E. (1993). Une représentation par automate fini des textes et des propriétés transformationnelles des verbes. *Lingvisticae Investigationes 17:1*, Amsterdam/Philadelphia : John Benjamins, 189-222.

SASTRE J. (2005). XML-Based Representation Formats of Local-Grammars for the NLP. *Proceedings of the 2nd Language & Technology Conference*, Poznan, Poland.

SKONNARD A., GUDGIN M. (2001). *Essential XML Quick Reference: A Programmer's Reference to XML, XPath, XSLT, XML Schema, SOAP and More*. Addison-Wesley Pub Co, 1st Edition, ISBN 021740958.

ULLMAN J. (1979). *Principles of Database Systems*. Computer Science Press, Rockville: Meryland.

VLIST E. (2002). *XML-Schéma*. Paris: O'Reilly. ISBN 2841772152.