Developing Language Resources and Applications with GEPPETTO

Fabio Ciravegna, Alberto Lavelli, Daniela Petrelli, Fabio Pianesi

Istituto per la Ricerca Scientifica e Tecnologica
Loc. Panté di Povo
I-38050 Trento, Italy
e-mail: {cirave|lavelli|petrelli|pianesi}@irst.itc.it

Abstract

In the development of LE applications there are two crucial areas: architectural design and resource development. In this paper we describe the design phase, current status, and future directions of a development environment, called GEPPETTO, whose aim is to address the two issues within the same environment. The design of GEPPETTO has been carried out using a user-centered approach, to secure the satisfaction of user needs and desiderata. The final aim of the environment is very complex and expensive; to obtain short-term intermediate results we adopted an incremental approach, developing firstly the tasks we considered more urgent and critical in LE applications and postponing the development of some other tasks. Currently only the environment for linguistic resource development has been fully implemented, whereas the other parts of the system are in progress. A case study of application of the currently implemented version of GEPPETTO to a system for Information Extraction from text is also presented; such case study confirmed the usefulness of the tools already developed and provided interesting suggestions concerning what is needed during the development of LE applications.

1. Introduction

The increasing diffusion of Linguistic Engineering (LE) applications has risen the need for tools capable of assisting the developers in their work. In this respect, it is possible to individuate two crucial areas: one is concerned with the development, debugging, etc. of the processors and the resources, and the other with the prototyping, debugging and validation of the architecture for the LE application. Some systems already make one of these two functionalities available; however, it is difficult to find a single system providing for both, despite the clear advantages of having them together within the same platform. In this paper we describe the design phase, current status, and future directions of a development environment, called GEPPETTO, whose aim is to address the two issues together.

Besides the above mentioned situation, another goal motivated our efforts on GEPPETTO: the need for a development system which meets user requirements, and provides for easy interactions and friendly working modalities (by *users*, we mean industrial as well as academic parties interested in developing real LE applications). Despite the clear importance of this point, it is difficult to evaluate present systems in this respect; the main reason is that most of them have not been designed with such a goal in mind, but are better seen as experimental tools, which address the needs and expectations of the research and academic community.

The design and implementation of an environment able to support each of the prototypical figures and functionalities involved in the development of LE applications is obviously a very complex and expensive task. Hence, we decided to deal with such effort envisioning some successive steps of development. Initially, our effort concentrated on what we consider particularly relevant, i.e. the area of linguistic resources (grammars and lexica), and the processors that use such resources: we carried out a further design phase specifically devoted to these functionalities and, after that, developed the environment that implements them.

Currently, GEPPETTO implements the results of this first step of design and implementation. Successive steps will include further phases of development and implementation of the other parts of the environment, finally addressing both the issues mentioned above (i.e., architectural design and resource development).

In the next section we summarize the outcomes of the global design phase; then, we move to the specification of the tools for the development of linguistic resources and processors (Section 3). Section 4 describes a case study where we applied the results described in Section 2 to the development of an application of Information Extraction from text; such case study showed some interesting directions for future developments. Finally, we outline future developments and further work on GEPPETTO.

2. Users, Tasks and LE Systems

To address the issue of user desiderata, the design phase of GEPPETTO exploited a particular "user-centered" approach, the so-called Participatory Design (PD) methodology. Within PD end-users act as fully empowered participants in the working group, sharing decisions together with the design team in all design phases. PD is highly effective in securing that the final system meets the needs of the end users. Further details on the specific techniques employed can be found in (Ciravegna et al., 1997a). PD made it possible to:

- provide an analysis of the development cycle of LE applications to serve as a foundation for the successive system design;
- individuate a number of prototypical figures representing the various actors, skills and competences involved in such a process: Linguistic Engineer (LER), Resource Manager (RM), and Processor Manager (PM);
- specify the needs and desiderata of each prototypical user and envision the corresponding functionalities: facilities for choosing among existing resources; specialized graphical browsers and editors for data; facilities for interactively testing and debugging processors, data, and the whole final architecture; facilities for providing the final delivery system;

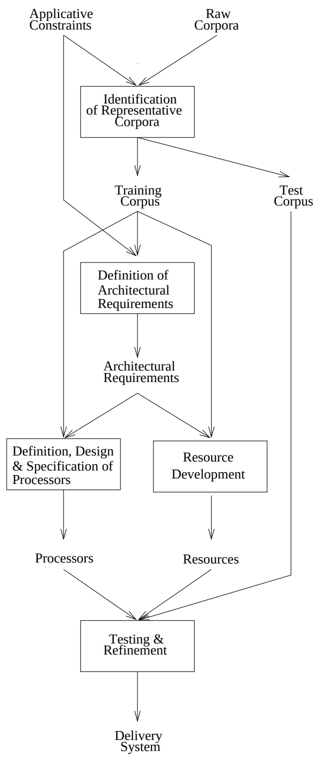
Applicative Constraints
Raw Corpora
Identification of Representative Corpora
Training Corpus
Test Corpus
Definition of Architectural Requirements
Architectural Requirements
Definition, Design & Specification of Processors
Resource Development
Processors
Resources
Testing & Refinement
Delivery System

Figure 1: Development cycle of LE applications

- test the resulting system with respect to the compliance with those needs and desiderata.

The discussion inside the working group was organized around three main (sequential) topics:

- the construction process of LE applications: development cycle, involved skills and tasks, etc.;
- actor desiderata: need of rapid prototyping, types of interfaces (graphical vs text-based), kind of openness of the architecture, availability of delivery systems, necessity of a modular approach to resource development, etc.;
- specifications of system facilities: kind of tools for browsing and editing data, capabilities of the API for integrating external resources, etc.

2.1. Building LE Applications

The working group focused on the abstract definition of the development cycle of LE applications and of the typologies of the involved users. As a matter of fact this is a requirement for a true LE approach to NLP systems.

The development cycle of LE applications was defined as in figure 1.

As a first step, *applicative constraints* must be considered. In fact, the working context of a LE application determines not only the global behavior of the LE application, but also the way the different modules interact to produce the desired behavior. Another preliminary step is the collection of *raw corpora*.

After such a preparatory work, the following development cycle typically takes place:

- *identification of representative corpora.* In this step the aforementioned raw corpora are classified and filtered to find a set of examples that is representative of the characteristics of the whole corpus. The resulting corpus is then split in two parts: one to be used during the system development (*training corpus*), the other during the testing phase (*test corpus*);
- *definition of the architectural requirements.* Given the applicative constraints and the characteristics of the corpus, the specific requirements of the LE application are defined;
- *definition, design and implementation of the processors,* according to the requirements of the previous point;
- *development of the resources,* according to the requirements arising from the previous analysis;
- *testing and refinement* of both the processors and the data collection.

Once all these steps have been gone through, the resulting architecture is delivered (*delivery system*).

The working group singled out three different user typologies which play a role in the tasks above. Each of them corresponds to different backgrounds, knowledge and skills:[1]

- Linguistic EngineeR (LER): expert on architectures for LE. Background: computer science; knowledge of computational linguistics;
- Resource Manager (RM): expert on data development (e.g., linguistic resources, knowledge bases). Background: e.g., computational linguistics, knowledge engineering; little knowledge of computer science;
- Processor Manager (PM): expert on processors for language processing. Background: computer science; knowledge of computational linguistics.

Accordingly, the development cycle has been refined as follows:

- identification of representative corpora: LER interacts with RM to provide a representative corpus for the application;
- definition of architectural requirements: given the corpus and the requirements for processors and resources, LER interacts with PM and RM to define the correct architecture;
- definition, design and implementation of the processors: PM chooses (or designs and implements) them;
- development of resources: RM chooses (or designs and implements) them;

[1] Actually the working group added also an Application Manager, i.e. an expert of the domains and of the users of the LE application. Such a figure is not discussed in this paper.

- test and refinement: LER checks the correspondence between the current implementation and the architectural requirements; the processors are tested by PM and the data collection by RM.

In the end, the working group had effectively specified the actions, the tasks, and the skills required to create LE applications.

The next phase was a further step of PD methodology applied to the design of the environment for the development of linguistic resources, which is described in the next subsection.

2.2. User Needs and Desiderata

The working group discussed some of the desirable features of a LE application development system, from the point of view of the users (i.e. the developers). Results can be summarized as follows:

- facilities for the rapid prototyping of LE applications via graphical interfaces;
- facilities for testing and debugging the whole architecture against test suites;
- aids for producing the delivery system;
- facilities for interactively testing and debugging processors and resources;
- specialized (graphical) browsers and editors for different kinds of resources;
- facilities for choosing among resources (e.g. lexica and grammars) provided by libraries already developed within GEPPETTO;
- facilities for integrating knowledge stored in external modules;
- facilities for integrating processors and resources different from those already provided by the environment.

One of the main outcomes of PD discussions was that the different users would benefit from a single, common tool capable of facilitating and supporting their mutual interactions (even when performing their tasks independently) as well as the integration of resources developed independently.[2]

On the other hand, given the different profiles and skills involved, each of the three user typologies needs different facilities and might prefer different interaction modalities. For example CLs tend to favor graphical interfaces that hide as much as possible low-level details (e.g. internal data representation). On the other hand, PMs have to cope with low level details. As it turns out, the ideal environment should both address the differing interaction styles of each user, and, at the same time, provide a uniform environment where their contributions can be easily integrated. These results can be obtained if, at any time, the user can select all and only the functionalities he/she actually needs.

A similar tension involves also linguistic data and processors. LERs want to see them as units that can be assembled to build the final architecture. PMs are inclined to consider the linguistic data as a unit, but see the processors as complex modules to manipulate. Finally, CLs obviously must be able to single out pieces of linguistic data and organize them in a significant way, while using the processors as black boxes.

3. Implementing the Methodology

As already said, we wanted GEPPETTO to provide support both for the task of manipulating resources, and for that of designing and testing LE architectures. Concerning the latter point, the working group made it clear that the user expects a system wherein he/she can choose different kinds of objects (processors, complex computational modules, and resources) by means of graphical facilities, assemble them, specify the relationships between the various parts, and assess the result. Once the global framework was established, the problem of methodology implementation in a real system arose. We had to choose between proceeding either in a breadth first manner, by continuing to specify the functionalities and tools for all the tasks at the same time and then implement them, or in depth first manner, e.g. by first addressing some of the tasks, and only afterwards considering the rest of the process. We chose the second option since it had the advantage of providing a usable system in a shorter time, albeit one with limited capabilities. We decided to address the problem of resource development, and in particular that of linguistic data (lexica and grammars), leaving other tasks (such as architecture design and development - but also the development of other kind of resources such as Knowledge Bases) as underspecified. This choice was motivated by the fact that among RMs, Computational Linguists have generally limited knowledge of computer science and then they particularly need user-friendly environments to operate with, while LERs and PMs and other types of RMs, being computer scientists, can more easily cope with a powerful API. As a consequence, concerning the architecture, we designed and implemented an API, which could provide some of the core functionalities of the architectural design task and concentrated on linguistic resource development.

PD discussions started again for making the requirements more detailed. The main outcomes of the first, more general design phase were confirmed, and some of the prototypical figures and functionalities became more specific. For instance, in the case of linguistic development the role of RM is embodied in that of a Computational Linguist (CL).

The rest of this paper is devoted to describe the present status of GEPPETTO. More precisely, in this section we will discuss the specifications and facilities for linguistic

[2] In this paper we focus on the interactions among users belonging to the different typologies and on the integration of their work. We will not address the important question of how to support the interactions and integration involving users of the same typology. For instance, we will not discuss here the issue of how the development of large grammars by different CLs can be properly supported by a development system for LE applications.

data manipulation and briefly introduce the API. The role of the latter will be better addressed in the following section where we briefly report on our experience on the use of GEPPETTO for building the architecture of an Information Extraction system, drawing some conclusion and hinting at future developments.

3.1. The Current GEPPETTO Environment

As said above, currently we have implemented only the part of the development environment which deals with linguistic data, while the architectural design is still accomplished via an API. In this section some of the relevant characteristics of GEPPETTO are outlined. A more detailed description of the current implementation of GEPPETTO is contained in (Ciravegna et al., 1997b).

In GEPPETTO the linguistic part of an application consists of two main elements: a *Linguistic Architecture* (i.e., a set of processors together with their mutual connections), and a *Linguistic System*. The latter is the collection of all the sets of linguistic descriptions relevant for the characterization of a given corpus. Given the kind of formalism adopted, namely TFL (see Section 3.2 below), a Linguistic System consists of: a type hierarchy, a grammar, a lexicon, and a set of macros. The concept of linguistic system is not simply conceived as a set of the different components, but it is a complex object with a central role in GEPPETTO: much of the development of LE applications is centered around linguistic systems, and also the graphical user interface has been designed taking into account the central role of such a notion. CLs edit, browse, and update linguistic systems; they can reuse existing linguistic systems, or parts thereof, to produce new ones.

GEPPETTO maintains a conceptual distinction between browsing/editing and testing/debugging. Actually, browsing/editing can be performed independently by different users, whereas testing/debugging can require a strict cooperation between different typology of users (e.g., the testing/debugging of a grammar require the use of a parser). This need emphasizes the advantage of a single environment for the whole development cycle: different users have dedicated facilities for development, but a common environment for integrating and testing. This is because an error can be due also to unexpected interactions between data and processors.

Before discussing how user needs have been implemented in GEPPETTO, we briefly introduce the formalism for linguistic data as it was developed by the CLs of the working group.

3.2. The Formalism for Linguistic Data

CLs participating in the working group suggested a formalism based on Typed Feature Logic (Carpenter, 1992). The reasons were as follows:

- TFL formalisms provide a way for breaking down the structure of linguistic data, allowing for a clear separation between the description of abstract linguistic types and that of grammatical rules and lexical entries. This facilitates knowledge encapsulation as well as a modular architecture of linguistic data. Such a modularity can play an important role in the reuse of existing data;
- typing secures to a high degree the consistency of the linguistic data. This speeds up the process of data editing and debugging;
- the formalism is well known and basic unification algorithms are available;
- it meets the demands of many current linguistic theories, e.g. LFG, GPSG, HPSG, etc.

TFL specifications are compiled into a graph format, where each node represents a Typed Feature Structure (TFS). Types and the type hierarchy have been implemented by adapting the encoding schema proposed by (Ait-Kaci et al., 1989) to the TFL format. This permits to efficiently handle very large type hierarchies as well as to account in a straightforward way for type disjunction. The standard TFL formalism has been modified to accommodate:

- *Declaration statements* specifying, for instance, that a certain object is not an ordinary TFS. In case its properties must be assessed by other, possibly external, modules; such a fact can be specified by means of *external constraints*;
- *External constraints* providing explicit links to external modules, e.g. morphological processors, independent KBs, etc.;
- *Directives* for the unifier. For instance, it is possible to force the unifier to consider in the first place the paths that have been observed to cause more frequent failures (Uszkoreit, 1991).
- *Macros*.

Declaration statements and external constraints improve the modularity and portability of the LE applications developed by means of GEPPETTO, by allowing the reuse of existing processors and/or data.

We now turn to a discussion of the facilities and tools provided to the different users.

3.3. Supporting Application Development

The study of architectural requirements is the task accomplished by LER and PM together. They also control the compliance of the LE application with the initial requirements and identify the processors that can satisfy the architectural requirements. As for the former, GEPPETTO provides support for: (a) the rapid prototyping of architectures by assembling already existing processors and linguistic systems, and (b) tests against a test corpus. Both data and processors are seen by the LER as black boxes that can be combined by means of a graphical interface.

As for the latter, they can choose among the processors made available by GEPPETTO or link external ones to the environment. Once a new processor has been properly linked via API, it is completely identical to the already available processors: it can be selected via the graphical interface, it can take advantage of the debugging/testing facilities, and so on.

Via API, it is also possible to interface LE applications with other kinds of external modules, e.g. modules which make available functionalities currently not provided by the environment and that can not be directly integrated (e.g. Knowledge Bases or morphological analyzers).

Initially, GEPPETTO featured two chart-based parsers (a bidirectional Head-Driven Bottom-Up (Satta and Stock, 1989) and a CYK-like) and a Head-Driven Bottom-Up non-deterministic generator (Pianesi. 1993). In the last months, GEPPETTO has been enhanced with tools based on Finite State technologies, particularly suited for one of our applications, i.e. the one which involves Information Extraction from text (Ciravegna et al., 1997c).

Once the architecture meets the requirements, a *delivery system* can be produced. It contains the selected linguistic system and processor(s), and does not include the graphical part of the GEPPETTO development environment.

3.4. Supporting Resource Development

A considerable effort has been devoted to create suitable (specialized) graphical tools for CLs. Recall that CL main task is to build a linguistic system satisfying the application requirements. The graphical environment must allow CL to ignore low-level details as much as possible, and concentrate on the linguistic aspects of data description.

CLs can build a linguistic system both by pasting already existing components (and modifying them when necessary) and by building it from scratch.[3]

As the data forming the parts of a linguistic system differ in attributes, global organization and functions, specialized graphical tools have been designed for browsing/editing the type hierarchy, the grammar, the lexicon and the macros.

The main tools for the CL are:

- a grapher for browsing and editing the type inheritance hierarchy. It displays mouse sensible nodes and allows to add/delete/modify nodes, as well as to modify the hierarchy itself;
- browsers for data sets such as lexicon, grammar and macros. They allow to add/delete/modify/copy elements in the data sets, as well as to undertake actions on the data set as a whole (e.g. compiling it);
- editors for editing and modifying properties of single lexical entries, grammar rules, macros and type hierarchy nodes. They include editors for TFL descriptions, feature appropriateness statements, etc. TFL-syntax error checking, TFL description compilation and TFS visualization are supported. Documentation and comment notes can be attached to each item;
- interactive and post-processing debugging tools (at now mainly a sophisticated chart browser).

[3] Currently GEPPETTO provides some standard linguistic systems for Italian.

Facilities are also supplied for computing statistics about performances on test suites. In particular, it is possible to detect points where unification failures arise. Such results can be exploited either to hand-tune the linguistic systems to the corpus needs, or by feeding them into a module which forces unification algorithms to consider unification failure hypothesis first, this way speeding up the whole processing.

3.5. GEPPETTO Evaluation

The implemented system (particularly the tools for the CLs) was assessed by means of an evaluation, to test its general usability and the quality of the proposed solutions. The results of the evaluation are summarized in the following; further details can be found in (Ciravegna et al., 1997a).

The experiments showed that the choices done and implemented into GEPPETTO effectively supported naive users in moving around and acting in a complex and unfamiliar environment. Even participants who had not read the manual and had only a little experience in NLP were able to complete their tasks in less then one hour. Through observations and interviews it could be verified that participants reached a good understanding of the system and judged it positively.

The experiments allowed to identify some weaknesses in the interface design and to solve them with a limited revision of the graphical interface. Experiments demonstrated that the adoption of PD can bring intuitiveness also in the design of a complex LE application development system: even users without any experience with GEPPETTO and limited knowledge in NLP were able to easily understand the system organization and to effectively use its tools to accomplish non trivial tasks.

4. Towards Architecture Definition: A Case Study in Information Extraction

We followed the development cycle described in Section 2.1 in order to define an architecture for Information Extraction (IE). IE is the mapping of NL texts into predefined, structured representations, or *templates*, which represent an extract of key information from the original text (Gaizauskas et al., 1997). The definition of an IE application is well suited for such a development cycle as:

- the task is formally well defined;
- a corpus sample is usually available in advance together with the answer keys (e.g. 100 texts in the MUC competitions (Sundheim, 1995));
- pragmatically, an IE architecture is task-driven, i.e. its only motivation is that of extracting information from the texts; all the actors involved in the development have the same goal.

In the last years the approach to IE has been standardized, especially for what concerns evaluation. In particular the IE task was evaluated in MUC-6 (Sundheim, 1995) according to the following lines:

- **NE**: Named Entity Recognition: recognition and classification of names, places, etc.
- **TE**: Template Element Construction: addition of descriptive information to NE results.
- **ST**: Scenario Template Production: insertion of TE results into event scenarios.
- **CO**: Coreference Resolution: identification of identity relations between entities in texts.

This is currently considered a standard for every IE system. The evaluation methodology obviously constrains also the way the task is performed to the point that it has been possible to define a generic architecture for IE (Hobbs, 1993). For this reason the definition of an architecture for IE is a complex process with strong external constraints; this means that it is not possible - as in other LE fields - to design the architecture according to the classic paradigm based on syntax, semantics, etc. All the process must be strongly task-driven.

We used the current GEPPETTO environment to build an architecture for IE within the FACILE project, a project funded by the European Union for text classification and IE.

Even if the environment is far from being complete, GEPPETTO's design features allowed a simplification of the task of developing such an architecture. First of all the mentioned development methodology allowed a clean definition of the requirements for the architecture. The output of the process was a detailed high level design for an architecture with constraints associated to each module. For example the architecture definition step posed as constraints among others both the evaluation requirements and some external applicative constraints (such as speed of analysis). Another requirement come out during the design phase was the need of using a strong syntactic component, mainly due to the characteristics of the input to be analyzed (Ciravegna et al., 1997c). PMs and RMs also established the precise constraints for each module and corresponding resources: concerning the modules, it was decided that all the architecture was designed as cascades of Finite State Transducers.

During implementation GEPPETTO mainly provided its default module for FSTs, including tracers, browsers and editors and the unifier. The FST modules were initially meant for syntactic analysis only, but turned out to be useful for all the levels of the architecture (e.g. also for semantic analysis, coreference resolution, and template filling). The graphical interface was used by the CL for developing the linguistic system. The API was used for connecting modules external to the current environment, i.e:

- the FACILE preprocessor (Black et al., forthcoming): including the Xerox Tools for morphological analysis and Part of Speech Tagging, a proper name and date recognizer, etc.;
- a knowledge representation tool;
- the scorer (currently the MUC-scorer 3.1).

Then:

- LER was mainly supported by the definition of the development cycle of the application and by the available architectures for playing with; it was possible to test different configurations making use of the already available modules;
- PM was supported by the presence of the available modules; the API allowed the introduction of new modules;
- RMs:
  - Computational Linguists were supported by the graphical interface for the development of lexica and the grammars;
  - no direct support was provided for knowledge engineers.

The resulting IE architecture was tested on a corpus of agency news in the domain of bond issues. On a corpus of 33 news it scored .72 in terms of recall and .74 in terms of precision. It currently works for Italian only, but an English version is under development within the FACILE project; a Russian version is also under development at IRST.

The IE application was a good test field for GEPPETTO as it stressed the task-driven architecture design and implementation that are among the main features motivating the project. The experience showed that the organization around a unique environment integrating tools for resource, module and architecture development is very useful, especially when providing different interaction modalities for different user typologies. The current GEPPETTO environment showed to be useful for the part currently covered (grammar and lexical development), but it needs hard working in the direction of architecture development, as the API alone is not friendly enough. The main shortcomings were found in an assumption that was implicit in the environment organization, i.e. the idea that an architecture is organized around the classic paradigm of a parser, a semantic analyzer, etc. As mentioned above an IE architecture is not necessarily organized in this way; to some extent this fact also constrained some choices during the architecture design phase. In the future we will investigate how it is possible to change the GEPPETTO organization in order to overcome such constraint.

5. Conclusions

In this paper we have described the design phase and current status of an environment whose aim is that of addressing two crucial issues: the development, debugging, etc. of processors and resources for LE applications, and the prototyping, debugging and validation of the architecture for the applications.

The currently implemented part of the environment was assessed through empirical evaluations with end-users, which produced positive results.

Furthermore, GEPPETTO has been extensively used, and is currently used, in a number of projects producing as diverse applications as text generation systems (GIST/LRE), information extraction systems (FACILE/LE), dialogue systems (TAMIC-P/LE).

The experience within FACILE is particularly relevant: on one hand, it confirmed the choices already implemented; on the other hand, it provided further suggestions on the correct way to proceed towards the goal of an integrated environment including also the architectural design of LE applications.

The version of GEPPETTO currently available runs in Allegro Common Lisp on Sun workstations; the graphical interface of the development environment requires CLIM and GRASPER.

References

Hassan Ait-Kaci, Robert Boyer, Patrick Lincoln, and Roger Nasr (1989). Efficient implementation of lattice operations. *ACM Transactions on Programming Languages and Systems*, 11(1):115–146.

William J. Black, Fabio Rinaldi, and David Mowatt. (forthcoming). Description of the NE system as used for MUC-7. In *Proceedings of the Seventh Message Understanding Conference (MUC-7)*.

B. Carpenter (1992). *The Logic of Typed Feature Structures*. Cambridge University Press, Cambridge, Massachusetts.

Fabio Ciravegna, Alberto Lavelli, Daniela Petrelli, and Fabio Pianesi (1997a). Participatory Design for Linguistic Engineering: the case of the GEPPETTO Development Environment. In *Proceedings of the ACL-EACL'97 Workshop on Computational Environments for Grammar Development and Linguistic Engineering*, Madrid, Spain, July.

Fabio Ciravegna, Alberto Lavelli, Daniela Petrelli, and Fabio Pianesi (1997b). The GEPPETTO Development Environment. Version 2.1. User Manual. Technical Report 9711-15, IRST, November.

Fabio Ciravegna, Alberto Lavelli, and Giorgio Satta (1997c). Efficient full parsing for information extraction. In *Proceedings of the Meeting of the Working Groups on Automatic Learning and Natural Language of the Associazione Italiana per l'Intelligenza Artificiale (AI*IA)*, Torino, November.

Robert Gaizauskas, Kevin Humpreys, Saliha Azzam, and Yorick Wilks (1997). Concepticons vs. lexicons: An architecture for multilingual information extraction. In Maria Teresa Pazienza, editor, *Information Extraction: A multidisciplinary approach to an emerging information technology*, pages 28–43. Springer Verlag.

Jerry R. Hobbs (1993). The generic information extraction system. In B. Sundheim, editor, *Fifth Message Understanding Conference (MUC-5)*, San Francisco, CA, August. Morgan Kaufmann Publishers, Inc.

Fabio Pianesi (1993). Head-driven bottom-up generation and Government and Binding: a unified perspective. In Helmut Horacek and Michael Zock, editors, *New Concepts in Natural Language Generation: Planning, Realization and Systems*, pages 187 – 214. Pinter Publishers, London.

Giorgio Satta and Oliviero Stock (1989). Formal properties and implementation of bidirectional charts. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI.

Beth Sundheim, editor (1995). *Sixth Message Understanding Conference (MUC-6)*. Morgan Kaufmann Publishers, Inc. S. Francisco (CA).

Hans Uszkoreit (1991). Strategies for adding control information to declarative grammars. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 237–245, Berkeley, California, USA.