SiSSA - An Infrastructure for NLP Application Development

A. Lavelli and **F. Pianesi**
ITC-irst
via Sommarive 18
38050 Povo (TN) ITALY
lavelli@itc.it
pianesi@itc.it

E. Maci and **I. Prodanof**
Ist. di Ling. Comput.
Area di Ricerca CNR
via Alfieri 1, San Cataldo
56010 Pisa ITALY
maci@ilc.pi.cnr.it
irina@ilc.pi.cnr.it

L. Dini and **G. Mazzini**
CELI
corso Moncalieri 21
10131 Torino ITALY
dini@celi.it
mazzini@celi.it

Abstract

Recently there has been a growing interest in infrastructures for sharing NLP tools and resources. This paper presents SiSSA, a project that aims at developing an infrastructure for prototyping, editing and validation of NLP application architectures. The system will provide the user with a graphical environment for (1) selecting the NLP activities relevant for the particular NLP task and the associated linguistic processors that execute them; (2) connecting new linguistic processors to SiSSA; (3) checking that the chosen architectural hypothesis corresponds to the functional specifications of the given application.

1 Introduction

In recent years there has been a growing interest in the commercial deployment of NLP technologies and in infrastructures for sharing NLP tools and resources. Such interest makes more and more urgent the availability of toolsets that allow an easy and quick integration of linguistic resources and modules and the rapid prototyping of NLP applications. An example of the efforts in such a direction is GATE (a General Architecture for Text Engineering, (Cunningham et al., 1997)), which provides a software infrastructure on top of which heterogeneous NLP processing modules may be evaluated and refined individually, or may be combined into larger application systems.

This paper presents SiSSA (Sistema integrato di Supporto allo Sviluppo di Applicazioni - Integrated System of Support to Application Development), a project with a twofold aim:

- the definition of a common metaformalism (called FIST) for the unification of different formalisms for grammar description, and the implementation of a Grammar Repository for storing grammars written using FIST;
- the implementation of an infrastructure for the rapid prototyping and testing of architectures for NLP systems, starting from linguistic processors made available by SiSSA itself.

In this paper we concentrate on the latter aspect, i.e. the infrastructure for designing NLP architectures. To this end, SiSSA provides the user with a graphical environment for (1) selecting the *linguistic activities* which are relevant to the particular application at hand, along with the linguistic processors that execute them; (2) checking that the chosen architectural hypothesis corresponds to the functional specifications of the application; (3) connecting to SiSSA new linguistic processors, this way making them available for the prototyping/design activities.

Thus, the design of the architecture of an NLP system amounts to a) identifying a sequence of linguistic activities to be performed; b) connecting them in a specific processing chain; and c) associating each linguistic activity to a suitable processor, selected among those made available by SiSSA. The term *project* is used to refer to the

product of the user's activity, namely, the architecture of the NLP application the user is building. A project encodes processing flows among basic units, each consisting of a linguistic task that is executed by a linguistic processor.

Projects have two uses. First, they store the status of a user session. Second, they are the main units of runtime modules: the SiSSA Manager (see below) *interprets* projects by executing the procedures chosen by the user and applying them to the document selected for execution.

SiSSA makes crucial use of state-of-the-art software technologies (CORBA, XML) in order to integrate the various modules in an effective way. The core of SiSSA has been developed in Java; hence it can run both on Windows and on Unix platforms.

SiSSA consists of two parts: an autonomous application (called *SiSSA Manager*) and a set of executable modules, henceforth called *processors*. The SiSSA Manager provides an infrastructure for architecture composition and processor integration. That is, it provides all the necessary support to allow the user to select linguistic activities, connect them in an overall processing chain, and associate each activity to a linguistic processor. Moreover, it takes care of executing the processing flow encoded in the project, reporting results to the user, etc.

A major goal while designing SiSSA was to allow the system to reuse existing processors as much as possible. This was meant to extend to processors located in other sites than the user's. Already existing processors are written in different programming languages and run on different hardware and software platforms; so this objective required the adoption of a distributed architecture, providing:

- flexibility (processors can be developed and updated independently);
- expandibility (new processors can be added);
- independence from the programming languages employed to implement the processors;
- distribution of execution on different hardware platforms.

As a result, the user can exploit for his/her needs processors that are located anywhere, provided that they have been *notified* to SiSSA, and enclosed in a wrapper so as to comply with the SiSSA interface (see Section 2.2 and 2.3).

In the following sections, we first present a detailed description of SiSSA. Then some considerations on the pratical use of the system are introduced. Finally some details about the current status of the SiSSA implementation and the future work follow.

2 SiSSA

The SiSSA system consists of:

- the SiSSA Manager;
- the processors;
- the grammars contained in the Grammar Repository;[1]
- formal specifications of the interfaces each processor has to provide in order to be "integrable" in SiSSA (this part uses CORBA, Common Object Request Broker Architecture - `http://www.corba.org`,);
- protocols for communication and formats for representation and exchange of information (achieved using XML (Bray et al., 2000)).

The difference between the third and the fourth element above is that the CORBA part specifies the details of the communication process without any reference to the linguistic characteristics of the integrable processors (this part could be largely reused in other non linguistic projects involving a distributed architecture); the specifically linguistic details are embedded in the XML documents passed between the processors.

2.1 SiSSA Architecture

The central element in the SiSSA architecture is an autonomous application, called *SiSSA Manager*. It is autonomous since it takes the initiative in the management of the processing flow of the SiSSA system, where it mainly plays the role of client. Its main tasks are the following:

[1] This part is not dealt with in the paper.

- to interact with the *Processor Repository* (the place where information about processors known to SiSSA is stored) to take a census, activate and connect the processors notified to the system;
- to present the system functionalities to the user by means of a web-based graphical interface. To this end, the SiSSA Manager acts as a server with respect to the processors towards which it mediates the "centralized" GUI. Through the latter, the SiSSA Manager not only interprets the user's actions but also gives her/him a report on the ongoing processing, storing and presenting logs and status messages coming from the active processors;
- to manage and interpret the projects built by the user.

The Processor Repository classifies the processors, by associating each of them to the appropriate class of linguistic processors (e.g., morphological analyzers, PoS taggers, etc.).[2] The Processor Repository also provides functionalities for permanently storing the properties associated with the processors registered in the repository. Among them, the properties that specify the methods for activating a processor are crucial. As a matter of fact, the single processors must be active in order to be available for use by SiSSA. The activation of a processor takes place by means of an *Activation Server*[3] reachable via CORBA at the URL stored in the Processor Repository and specifying the corresponding *activation string*. The information is stored in the repository using RDF[4] and RDFS[5] (Resource Description Framework and RDF Schema). RDF Schema makes available tools to check that the descriptions of the processors' characteristics comply with SiSSA Manager's constraints. The RDF specification of the processors made available in SiSSA is usually built using a graphical interface. The adoption of RDF and RDF Schema enhances the generality of SiSSA, by avoiding *ad hoc* languages for resource description, and *ad hoc* schemas for the validation of documents describing the processors.

Turning to the processors stored in the repository, they mainly play the role of servers which are activated upon request by the SiSSA Manager.

The goal of making available distributed architectures for projects is pursued through the adoption of CORBA (Common Object Request Broker Architecture - `http://www.corba.org`, developed by the OMG industry consortium), which acts as the glue keeping together the executable parts of SiSSA.[6] To be available to SiSSA, processors must be registered in the Processor Repository. To this end, they must exhibit interfaces that comply with a set of specifications defined using the CORBA Interface Definition Language (IDL). Thus, providing the compliant interfaces is a necessary step towards integrating new processors within SiSSA.

As to communication formats, the overall goals of SiSSA made the adoption of XML (Bray et al., 2000) a natural choice. Thus messages are exchanged in the form of XML documents of type `process-data` (see Section 2.3). These documents incorporate in a single structure: the object to be processed, and information relevant for the processing itself (metadata). The generality of such a format permits its use both for the communication between the SiSSA Manager and the processors, and for those directly taking place among the processors.

2.2 Communication Protocols

As said, SiSSA provides a set of formal IDL specifications which the interfaces of processors aiming at being integrated in the environment must adhere to. Such specifications model the interaction between the SiSSA Manager and the proces-

[2] Currently the following classes of processors are defined in SiSSA: documentProc, preprocessorProc, textZonerProc, nERecognizerProc, morphologyProc, poSTaggerProc, syntaxProc, semanticsProc, DiscourseProc, XSLProc.

[3] In case the processor resides on a computer directly accessible to the SiSSA Manager, it can be activated by means of a shell command. In the following we always consider the case in which the activation server is needed.

[4] RDF is a W3C Recommendation of 22 February 1999 (Lassila and Swick, 1999) that specifies a declarative language (based on XML) formally equivalent to propositional logics. RDF is usually employed to describe resources on the web.

[5] RDF Schema Specification 1.0, published as a W3C Candidate Recommendation in March 2000 (Brickley and Guha, 2000).

[6] The SiSSA Manager uses ORBacus 3.3.2, `http://www.ooc.com/ob/`.

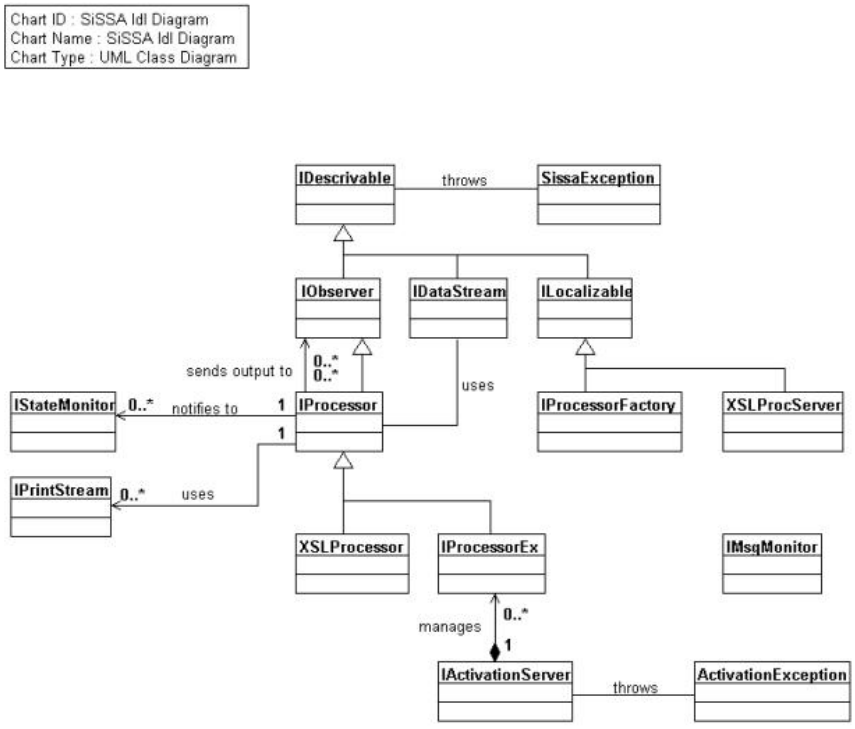

Figure 1: The UML diagram of the SiSSA IDLs.

sors, as mediated by CORBA. They can be seen as a contract that the SiSSA Manager and the processors have to comply for their mutual integration to be successful. A UML diagram of the SiSSA IDL specifications is shown in Figure 1.

The scenario of the cooperation between the SiSSA Manager and a generic processor can be described as follows:

- the processor's activation server starts and connects on the CORBA bus as a named server at a specified URL (i.e, the `corbaloc:` URL stored in the Processor Repository);
- the SiSSA Manager, in its turn connected to the CORBA bus, can contact the activation server using the `corbaloc:` URL specified in the Processor Repository; using the processor's activation string it can ask the server to activate the corresponding processor;
- from now on, the interaction takes place directly between the SiSSA Manager and the processors whose interface it obtained;
- the SiSSA Manager can in this way act as a true manager, establishing and removing the connections between the processors according to the design of the processing flow decided by the user.

In SiSSA, the communication is asynchronous, and is implemented by means of a flow of XML documents. The processors and the SiSSA Manager can be both the source and the target of communication. Moreover, each communication can have more than one target.

Being a possible target of communications, each registrable processor provides the functionalities of the interface `IObserver`. The SiSSA Manager's way to establish/remove the relationships between processors according to the user requirements amounts to inserting/deleting observers into a processor's list of observers.

Besides the communication related to the linguistic processing, other relevant communication flows concern error messages, and information tracing. Logs and messages directed to the user are managed through the interface `IMsgMonitor`. Finally, the interface

IStateMonitor (provided by the SiSSA Manager) allows each processor to signal its callers the status of its own processing (an example of its use is shown in the bottom bar of the window shown in Figure 3).

An important service provided by the SiSSA Manager is the XSL[7] processing of XML documents. To this end, the SiSSA Manager provides the interface XSLProcServer, through which XSLProcessor (a processor specialized in XSL transformations) is made available.[8] This feature allows the insertion of XSL transformations between any pair of processors, this way providing the possibility of adapting one processor's output to the requirement of the following one(s). This feature is of the utmost importance for augmenting SiSSA's capabilities of integrating and successfully making available a wide range of processors.

2.3 Communication and Representation Formats

Communications take place using a "data container" modeled by the interface IDataStream. An object that implements such interface is sent by a processor to each of its observers on completion of its processing.

IDataStream is designed as a container rather than as a structured model of the data exchanged. The definition of structured models for data is completely independent from IDataStream, and is obtained through different means. Indeed, given that the contents of data streams are XML documents, their structure is made explicit by means of Document Type Definitions (DTDs).

SiSSA is a development environment, meant to be open to the integration of new components, whereby the latter can differ among them along a number of dimension, including the input/output formats. At the same time, SiSSA should allow the user an adequate level of control over the intermediate results produced during the computation (i.e., the output of each processor). XML allows a representation of data which is transparent and accessible to the developer, without the need for her/him to know the details of the implementation of the single components. At the same time, it does not increase the complexity of the CORBA interfaces that encapsulate such data.

The data defined in XML are associated to a document of type process-data. Each document of type process-data necessarily includes two parts:

- linguistic data, usually corresponding to the result of the computation done by the source processor;
- metadata. Their role is to specify: the level of analysis accomplished by the source processor (e.g., tokenisation, parsing, etc.); the unique identifier of the processor originating the data; further useful information about processing (time of execution, rules applied, etc.). Moreover, metadata make available a unique identifier for the process-data document. This is useful so to associate the input with the different output structures produced by the different processing steps.

The linguistic data have to comply with the definitions specified for the different classes of processors. Such classes are identified by the attribute level-of-analysis present in the metadata (e.g., morphological analyzer, PoS tagger, chunk parser, etc.) and should take into account (at least to a certain extent) idiosyncrasies of specific processors. For instance, a morphological analyzer can adopt a set of category labels not entirely coincident with that of another morphological analyzer.

Obviously, a structure that aims to carry linguistic data of different nature, and so differently represented, can become quite complex when the levels of analysis taken into consideration increase. Moreover, during the development phase, the problem arises of the integration of data structures relative to levels of analysis previously not taken into consideration, as well as of data structures idiosyncratic to processors belonging to some classes. The modular nature of the DTDs for XML allows a neat distinction among metadata, and data relative to classes of processors (idiosyncratic data). The former are described in a

[7] The Extensible Style Language (XSL (Adler et al., 2000)) is a language that allows to transform data from one XML representation to another.

[8] The SiSSA Manager uses the Xalan XSLT processor, http://xml.apache.org.

single DTD, defined as part of the resources internal to SiSSA, while the latter can be conveyed by various DTDs, possibly made available in SiSSA along with each processor.

As said, each processor at the end of its processing makes available a document of type `process-data`, which contains exclusively the output data of the specific processor – and obviously the corresponding metadata. Such a document is a representation of the output of the processor that generated it, and does not contain any representation relative to previous levels of analysis, the input text or the history of the processing done so far. Thus, for efficiency reasons `process-data` are not incremental collection of all the data produced by the various processors.

At the same time, the need to keep a link between the input test and the output produced by the system cannot be ignored. It is also reasonable that in certain situations (e.g., during testing and debugging) the structures produced by the intermediate processors, as well as the metadata of the various processors, are needed to show or save tracing information. In the proposed architecture, this task is accomplished by the SiSSA Manager, that can register itself as an observer of any processor; in this way it can access the processor output and show it to the user or build a tracing structure.[9]

3 SiSSA at Work

There are two main activities regarding the characteristics of SiSSA described in this paper: the development of projects and the integration of processors.

3.1 Projects

The creation and editing of projects takes place exclusively via the SiSSA graphical interface. First the user decides which linguistic activities are relevant to her/his project. Then s/he can browse the Processor Repository, searching for those which are suitable to realize each linguistic activity.[10] Finally, s/he composes them into a project.

When it is necessary to test a given project on a text the SiSSA Manager prepares a suitable stream (`IDataStream`) and sends it to the processor selected as the first in the analysis chain. The processor interprets the metadata, executes the specified operation on the linguistic data and finally sends its output to all its observers; some of them can be required to perform further processing on the linguistic data. The output produced by a processor is sent to the SiSSA Manager as well, so that it can be shown to the user in a suitable form.

In Figure 2 the starting page of the SiSSA system is shown. In the upper part of the window there are a few buttons that are present in all the pages of SiSSA. From left to right:

- `Home`: a link to the starting page of SiSSA;
- `SiSSA Manager`: a link to the page of the SiSSA Manager;
- `Progetti` (projects): a link to the page that allows to create, edit, and activate the user's projects;
- `Repository`: a link to the page for interacting with the Processor Repository;
- `Help`: an online help.

Figure 3 shows the applet that interactively monitors the status of the project currently active and displays it to the user. In the upper part of the window the details of the active project are shown: the processors (left), the connections between processors (middle), and the XSL filters (right). In the lower part of the window the messages coming from the processors are shown. The bottom bar shows which of the processors/filters is currently active (using the `IStateMonitor` interface described in Section 2.2).

3.2 Integration of processors

Differently from the activity of creation and editing of projects, only the final part of the work involved in the integration of processors is accomplished via the SISSA graphical interface (more

[9] The SiSSA Manager uses the Xerces XML parser, `http://xml.apache.org`.

[10] The processors currently available within SiSSA are some processors developed by the partners of the project: the morphological analyzer and the parser of NLGRADE (ILC, written in C and running under Windows: (Prodanof et al., 1998; Prodanof et al., 2000)) and the preprocessor and the parser of GEPPETTO (ITC-irst, written in Common Lisp and running under Solaris: (Ciravegna et al., 1997; Ciravegna et al., 1998)).

Figure 2: The starting page of SiSSA.

precisely, the registration in the Processor Repository of the availability of the processors).

In order to make a processor SiSSA-compliant, the following steps are necessary:

- to provide it with a wrapper so that it communicates via the CORBA IDLs of SISSA;
- to make a translation between the processor's native input/output and the corresponding linguistic representation specified by `process-data`;
- to register the processor in the Processor Repository using the SiSSA graphical interface; during this step the class of the processor, its `corbaloc:` URL and activation string have to be specified.

4 Conclusions

The release 1.2 of the SiSSA Manager has been completed and is currently under use at the sites involved in the SiSSA project.

Given the emphasis on rapid prototyping, SiSSA has been developed with flexibility during the development phase as a primary goal. Obviously this flexibility is no longer needed when an application is delivered (on the contrary flexibility can considerably reduce the performances of the system). We are currently studying approaches to allow the delivery of efficient runtime processors.

Currently the SiSSA Manager is a single-user application. An extension is the possibility of having more than one user that uses it at the same time.

The SiSSA system has been developed as part of the TAL project. TAL is a project partially funded by the Italian Ministry for University and Scientific Research.

References

Sharon Adler, Anders Berglund, Jeff Caruso, Stephen Deach, Paul Grosso, Eduardo Gutentag, Alex Milowski, Scott Parnell, Jeremy Richman, and Steve Zilles. 2000. Extensible Stylesheet Language (XSL) Version 1.0. W3C Candidate Recommendation 21 November 2000. `http://www.w3.org/TR/xsl/`.

Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, and Eve Maler. 2000. Extensible Markup Language (XML) 1.0 (Second Edition). W3C Recommendation 6 October 2000. `http://www.w3.org/TR/REC-xml/`.

Dan Brickley and R.V. Guha. 2000. Resource Description Framework (RDF) Schema Specification 1.0. W3C Candidate Recommendation

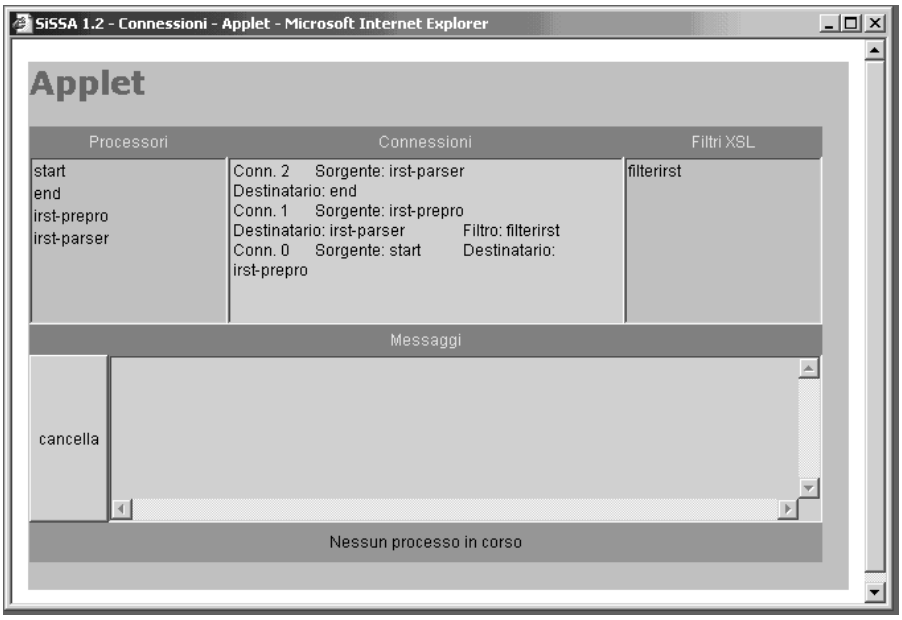

Figure 3: The applet that shows the connections between processors.

27 March 2000. `http://www.w3.org/TR/2000/CR-rdf-schema-20000327/`.

Fabio Ciravegna, Alberto Lavelli, Daniela Petrelli, and Fabio Pianesi. 1997. 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. 1998. Developing language resources and applications with GEPPETTO. In *Proceedings of the First International Conference on Language Resources & Evaluation*, Granada, Spain.

Hamish Cunningham, K. Humphreys, Robert Gaizauskas, and Yorick Wilks. 1997. Software infrastructure for natural language processing. In *Proceedings of the Fifth Conference on Applied Natural Language Processing (ANLP-97)*.

Ora Lassila and Ralph R. Swick. 1999. Resource Description Framework (RDF) Model and Syntax Specification. W3C Recommendation 22 February 1999. `http://www.w3.org/TR/REC-rdf-syntax/`.

I. Prodanof, A. Cappelli, L. Moretti, M. Carenini, P. Moreschini, and M. Vanocchi. 1998. A grammar development environment for reusable and easily customizable NL applications. In *Proceedings of the First International Conference on Language Resources & Evaluation*, Granada, Spain.

I. Prodanof, A. Cappelli, and L. Moretti. 2000. Reusability as easy adaptability: A substantial advance in NL technology. In *Proceedings of the Second International Conference on Language Resources & Evaluation*, Athens, Greece.