Software Infrastructure for Language Resources: a Taxonomy of Previous Work and a Requirements Analysis.

Hamish Cunningham, Kalina Bontcheva, Valentin Tablan, Yorick Wilks

Department of Computer Science and
Institute for Language, Speech and Hearing,
University of Sheffield, UK
{hamish,kalina,valyt,yorick}@dcs.shef.ac.uk

Abstract

This paper presents a taxonomy of previous work on infrastructures, architectures and development environments for representing and processing Language Resources (LRs), corpora, and annotations. This classification is then used to derive a set of requirements for a Software Architecture for Language Engineering (SALE). The analysis shows that a SALE should address common problems and support typical activities in the development, deployment, and maintenance of LE software. The results will be used in the next phase of construction of an infrastructure for LR production, distribution, and access.

1. Introduction

This paper reports work following from that discussed in the *Distributing and Accessing Language Resources* workshop at the previous conference in this series (LREC-1 1998; http://www.dcs.shef.ac.uk/~hamish/dalr/).
The paper addresses the provision of software infrastructure for research and development of language resources and language processing software. It presents a wide range of references to literature in this area, and a set of requirements that abstract from the work refered to. It is intended as a resource for workers in the area, and as a design step in the next phase of construction of an infrastructure for LR production, distribution and access. The paper begins with a description of what we mean by infrastructure, drawing on software engineering sources. Following a note on terminology relating to the common distinction between processing and data in language processing R&D, a review of previous work is presented as a stuctured set of references. This taxonomy is then used to organise a set of requirements for such infrastructures. The requirements are presented as use cases. The results will be used in the next phase of construction of an infrastructure for LR production, distribution, and access, GATE, a General Architecture for Text Engineering (Cunningham et al., 1999).

2. Software Infrastructure for Language Resources

The context of this work is the construction of software infrastructure for language processing: software that is intended to apply to whole families of problems within this field, and to be like a craftsman's toolbox in service of construction and experimentation. We consider three types of infrastructural systems: frameworks; architectures; development environments.

A *framework* typically means an object oriented class library that has been designed with a certain domain in mind, and which can be tailored and extended to solve problems in that domain. They may also be known as platforms, or component systems.

All software systems have an *architecture*. Sometimes the architecture is explicit, perhaps conforming to certain standards or patterns, sometimes it is implicit. Where an architecture is explicit and targetted on more than one system, it is known as a reference architecture, or a domain-specific architecture. The former is "a software architecture for a family of application systems." (Tracz, 1995) The term Domain Specific Software Architecture (DSSA – subject of an eponymous ARPA research programme (Hayes-Roth, 1994; Tracz, 1995)) "applies to architectures designed to address the known architectural abstractions specific to given problem domains." (Clements and Northrop, 1996)

An implementation of an architecture that includes some graphical tools for building and testing systems is a *development environment*. One of the benefits of an explicit and repeatable architecture is that a symbiotic relationship with a dedicated development environment can arise. In this relationship the development environment can help designers confrom to architectural principles and visualise the effect of various design choices and can provide code libraries tailored to the architecture.

3. Language Resources and Processing Resources

Like other software, LE programs consist of data and algorithms. The current trend in software development is to model both data and algorithms together, as *objects*. (Older development methods like Structured Analysis (Yourdon, 1989) kept them largely separate.) Systems that adopt the new approach are referred to as Object-Oriented (OO), and there are good reasons to believe that OO software is easier to build and maintain (Booch, 1994; Yourdon, 1996).

In the domain of human language processing R&D, however, the choice is not quite so clear cut. Language data, in various forms, is of such significance in the field that it is frequently worked on independently of the algorithms that process it. Such data has even come to have its own term, *Language Resources* (LRs) (LREC-1, 1998), covering many data sources, from lexicons to corpora.

In recognition of this distinction, we will adopt the following terminology:

Language Resource (LR): refers to data-only resources such as lexicons, corpora, thesauruses or ontologies. Some LRs come with software (e.g. Wordnet has both a user query interface and C and Prolog APIs), but where this is only a means of accessing the underlying data we'll still define such resources as LRs.

Processing Resource (PR): refers to resources whose character is principally programmatic or algorithmic, such as lemmatisers, generators, translators, parsers or speech recognisers. For example a part-of-speech tagger is best characterised by reference to the process it performs on text. PRs typically *include* LRs, e.g. a tagger often has a lexicon.

PRs can be viewed as algorithms that map between different types of LR, and which typically use LRs in the mapping process. An MT engine, for example, maps a monolingual corpus into a multilingual aligned corpus using lexicons, grammars, etc.

Adopting the PR/LR distinction is a matter of conforming to established domain practice and terminology. It does not imply that we cannot model the domain (or build software to support it) in an object-oriented manner; indeed the models developed for this work are themselves object-oriented.

The rest of the paper provides references to previous work in this area, structured taxonomically. This taxonomy is then used to organise a set of requirements for such infrastructures. The requirements are presented as a set of twenty-two use cases which encapsulate the desiderata for an ideal infrastructure for the support of Language Resource production, access and distribution.

4. Taxonomy

This part gives a brief review of the various approaches that have been taken to SALE (for a longer review see (Cunningham, 2000)). The prime criterion for consideration in this part is being *infrastructural*. In order to provide an organising principle for the discussion we begin in this chapter by extrapolating a set of architectural issues that represents the union of those addressed by the various researchers cited. This has the advantage of being easier to transform into software design and the disadvantage that multipurpose infrastructures appear in several categories. Cakes, eating them, etc. The breakdown of categories is as follows.

1. **Processing resources**
   (a) *Locating, loading and initialising components from local and non-local machines.*

   TalLab (Wolinski et al., 1998) Asynchronous agent architecture inc. component execution model.

   Corelli/Calypso (Zajac, 1998b) The Calypso Document Manager and component integration architecture.

   GATE (Cunningham et al., 1995; Cunningham et al., 1996c; Cunningham et al., 1996b; Cunningham et al., 1996d; Cunningham et al., 1996a; Cunningham et al., 1997b; Cunningham et al., 1997a; Cunningham et al., 1998b; Cunningham et al., 1998a; Cunningham, 1999; Cunningham et al., 1999; Gaizauskas et al., 1996) A General Architecture for Text Engineering.

   ICE (Amtrup, 1995) The INTARC Communication Environment for Verbmobil.

   TIPSTER (Grishman, 1997)

   (b) *Executing processing components, serially or in parallel.*

   Whiteboard architecture (Boitet and Seligman, 1994) A distributed whiteboard architecture with examples in a speech translation system.

   TalLab

   Pantome (Edmondson and Iles, 1994a; Edmondson and Iles, 1994b) Parallel architecture exemplified in a text-to-speech application.

   TALISMAN (Stefanini and Deamzeau, 1995; Koning et al., 1995) Distributed multi-agent architecture exemplified in syntactic analysis.

   ASL (von Hahn, 1994) A review of architectural problems focussing on the issue of linearity, and presenting the ASL architecture.

   XeLDA (Poirier, 1999) Architecture and framework for distributed collection of finite state transducers (FSTs).

   TIPSTER

   Corelli/Calypso

   GATE

   ICE

   (c) *Representing information about components.*

   Constraint-based interfacing (Busemann, 1999) Using a constraint language to describe processing components in order to facilitate integration.

   TIPSTER

   Corelli/Calypso

   GATE

   ICE

   Verbmobil (Görz et al., 1996; Bos et al., 1998) Inter-module interface language in Verbmobil.

   (d) *Factoring out commonalities amongst components.*

   Generic IE (Hobbs, 1993) Typical module set for an IE system.

   Tapestry (Cheong et al., 1994) A toolkit for building IE systems exemplified in the MFE IE system.

   NLI+ (Johnson and Rosenberg, 1995) Portability of NL interfaces to databases.

TARO (Ibrahim and Cummins, 1989) An OO syntactic analyser toolkit based on a specification language.

Reiter and Dale (Reiter, 1994; Reiter, 1999; Reiter and Dale, 1999; Reiter and Dale, 2000) From the perspective of applied Natural Language Generation work Reiter and later Dale review and categorise NLG components and systems. Reiter argues for descriptive properties of architectures vs. standardisation in (Reiter, 1999).

RAGS (Cahill et al., 1999b; Paiva, 1998; Cahill and Reape, 1998; Cahill et al., 1999a) Theory neutrality is harder to achieve in NLG than in NLU; the RAGS project.

Generic IR (TIPSTER, 1995) Typical module set for an IR system.

Architecture for IR (Cutting et al., 1991) An OO architecture for IR systems.

Spoken Dialogue architecture (LuperFoy et al., 1998) An architecture for spoken dialogue systems.

TIPSTER

2. **Language Resources, corpora and annotation**

(a) *Accessing data components.*

EUDICO (Brugman et al., 1998) The EUDICO distributed annotated corpora system.

Inquery **distributed IR** (Cahoon and McKinley, 1996) Making the Inquery IR system distributed.

W3-Corpora site (University of Essex, 1999) Searchable on-line annotated corpora.

Ontolingua (Fikes and Farquhar, 1999) Distributed reusable ontologies.

Ontology efficiency (Hendler and Stoffel, 1999) Efficient RDBMS backend for ontologies.

NLG Lexicon merging (Jing and McKeown, 1998) Merging large scale lexical resources inc. WordNet and Comlex for NLG.

GATE/DALR (Peters et al., 1998; Cunningham et al., 1998a) Distributing and accessing LRs.

(b) *Managing collections of documents (including recordings) and their formats.*

EUDICO

TIPSTER

Corelli/Calypso

GATE

HTK (Young et al., 1999) An architecture and framework for developing HMM-based Automatic Speech Recognition.

(c) *Representing information about text and speech.*

EUDICO

Shiraz MT Architecture (Amtrup, 1999) Chart- and unification-based architecture for MT.

TEI and CES (Sperberg-McQueen and Burnard, 1994; Ide, 1998a; Ide, 1998b; Ide and Priest-Dorman, 1999) Text Encoding Initiative and Corpus Encoding Standard.

SGML and XML (Goldfarb, 1990; Goldfarb and Prescod, 1998; Connolly, 1997; Nelson, 1997) Various sources on SGML and XML.

OODBs and SGML (Olson and Lee, 1997) Using Object databases for storing and retrieving SGML documents.

LT NSL and LT XML (Mikheev and Finch, Washington, DC, 1997; McKelvie et al., 1997; Isard et al., 1998; Brew et al., 1999; McKelvie et al., 1998) The Edinburgh SGML and XML library and tools.

THI (Thurmair, 1996) Thurmair's Text Handling Interface (based on SGML).

Corelli/Calypso

GATE

Annotation Graphs (Bird and Liberman, 1998; Bird and Liberman, 1999a; Bird and Liberman, 1999b; Bird et al., 2000) Annotation graph formalism, mainly applied to speech annotation.

HTK

(d) *Representing information about language.*

RAGS

Shiraz MT Architecture

Eurotra architecture (Schutz et al., 1991) An 'open and modular' architecture for MT promoting resource reuse.

GATE/DALR

TFw Terminology Framework (Fischer et al., 1996) Abstract model of Thesauri and terminology maintenance OO framework.

ARIES (Goni et al., 1997) Formalism and development tools for Spanish morphological lexicons.

FSTs and AVMs (Zajac, 1998a) Unified FST/AVM formalism for morphological lexicons.

GDEs (Netter and Pianesi, 1997; Estival et al., 1997) Grammar development in an LE context.

ALEP (Simkins, 1992; Simkins, 1994; Eriksson, 1996) Feature-structure based architecture and development environment.

AVM-based framework (Zajac, 1992) A framework for defining NLP systems based on AVMs.

HTK

(e) *Indexing and retrieving information.*

Generic IR

TIPSTER DN2 (Buckley, 1998) A communication protocol based on Z39.50 for detection interactions between querying application and search engine.

FireWorks (Hendry and Harper, 1996) A UI framework for building IR systems.

P-OQL (Henrich, 1996) IR extensions for the PCTE repository interface standard.

CUE (Mason, 1998) Indexing and search of annotated corpora.

W3-Corpora site

Corpus Query System (Christ, 1994; Christ, 1995) Indexing and search of corpora; linking with WordNet.

TIPSTER

3. **Methods and applications**
   (a) *Method support*
      i. *FST, unification and statistics over information.*

      CMU-Cambridge Stat Modelling toolkit (Clarkson and Rosenfeld, 1997) A set of command-line tools for building ngram language models.
      Shiraz MT Architecture
      HTK
      ii. *Comparing different versions of information.*

      DARPA (Sparck-Jones and Galliers, 1996) ATIS, MT, etc. etc.
      TEMAA (Paggio, 1998) Spelling and grammar checker evaluation, and general evaluation framework.
      ETE (Li et al., 1998) A GUI test environment for NLU systems.
      GATE
   (b) *Application issues*
      i. *Storing information.*

      Ontology efficiency (Hendler and Stoffel, 1999) Efficient RDBMS backend for ontologies.
      OODBs and SGML
      GATE
      HTK
      EUDICO
      ii. *Deployment and embedding* (executable programs; databases; libraries; components).

      GATE
      HTK
      XeLDA
   (c) *Development issues*
      i. *Interoperation with other infrastructures.*

      GATE
      ii. *Viewing and editing data components and information.*

      GATE
      iii. *UI access to architectural facilities (development environments).*

      InfoGrid (Rao et al., 1992) UI and interaction model framework for IR systems.
      FireWorks
      ETE
      GATE
      EUDICO
      HTK

5. Requirements analysis

As its name suggests, GATE is intended to be general, to cater for a wide range of LE activities and to encompass a large number of the useful infrastructural activities that have been identified by other work in this area. Given that language processing is still very much a research field, this task is potentially open-ended, and the question arises of how to restrict the endeavour to manageable proportions. The approach that we have taken is to make a distinction between software that aims to solve some research goal and software that provides an implementation of tasks that are common to a number of research goals (and that are not themselves active subjects of research). The latter are candidates for inclusion in the architecture; the former are not. In other words, GATE provides infrastructure tools and not research results: anything that is an open research topic is not properly part of the architecture. *How* to implement certain common algorithms or data structures may be part of the system; *what* algorithms and data structures to choose for a particular application or research project is not.

Exceptions to the 'no research' rule are made for two reasons. First, where a task is a valid research subject, but nonetheless so common amongst LE systems that a SALE will benefit greatly from the inclusion of a default implementation. Tokenisation of text is such a subject: whilst not a solved problem (particularly for languages such as Japanese or Chinese), still most researchers would prefer to take a simple tokenisation scheme as read. Secondly, SALEs must be developed *in context*, along with the processing systems that will use them. In our case the primary context has been Information Extraction research, and we have actively distributed IE components along with versions of GATE in order to promote both the architecture and work on IE itself. (Note that these components are actually separate from GATE itself.

A SALE should support all the activities involved in the development, deployment and maintenance of LE software. In particular, anything that represents common ground amongst LE applications (i.e. gets implemented regularly) and anything done by support tools that help LE workers is a candidate for desiderata for the architecture. We may identify roles for SALE in the development of LE applications, technologies, methods and components. Examples of these roles:

For applications, allow easy embedding in mainstream software architectures (e.g. by exploiting component-based devevelopment, Java, the Internet).

For technologies, provide measurement apparatus (e.g. precision and recall of IE outputs relative to manual annotation).

For methods, implement common algorithms (e.g. Baum-Welch for HMMs, FST over annotation) and support common data structures (e.g. annotation).

For components, provide abstractions that model their commonalities.

We can classify SALE roles according to the issues that have been addressed by previous work on LE infrastructure using the taxonomy of chapter 7. This has the advantage of being a reflection of infrastructure requirements that have been developed in close association with the client group, and it is this classification that we will use in this chapter, which presents a set of use cases encapsulating the requirement set for GATE.

What are 'use cases'? "In essence, a **use case** is a typical interaction between a user and a computer system" (Fowler and Scott, 1997), but beyond this statement there is no fixed definition of what they should look like ((Cockburn, 1997) cites experience of 18 variations). They are partly defined by their purpose, which is to identify what a computer system should do for its users. Fowler (Fowler and Scott, 2000) identifies these properties:

- "A use case captures some user-visible function.
- A use case may be small or large.
- A use case achieves a discrete goal for the user."

In keeping with our low-overhead development philosophy our use cases are natural language descriptions of the ways in which a SALE is used by its clients. Some of the use cases below stretch the definitions a little and might perhaps be called more accurately *use desiderata*; our belief is that development processes should bend to fit developers and their domains, not the other way around.

The clients of a SALE are the *actors* in use cases; they may be human (software developers, researchers, teachers or students) or software (the programs written and used by the human clients). The client set includes:

- expert programmers producing applications software that processes human language;
- non-expert programmers writing experimental software for research purposes;
- systems administrators supporting language researchers;
- non-programming language researchers performing experiments with software written by others;
- teachers of language technologies.

Section 5.1. gives general use cases; section 5.2. PR and LR use cases; section 5.3. use cases relating to methods; section 5.4. those specific to applications; section 5.5. those for development environments.

5.1. General desiderata

Use case 1: LE research and development

Goal summary To support LE R&D workers producing software and performing experiments.

Brief description During design, developers use the architectural component of SALE for guidance on the overall shape of the system. During development they use the framework for implementations of the architecture, and of commonly occuring tasks. The development environment is used for convenient ways of exploiting the framework and of accessing common tasks. For deployment the framework is available independently of the development environment and can be embedded in other applications.

Use case 2: Documentation, maintenance, and support

Goal summary To document, maintain and support the architecture.

Brief description Without adequate documentation of its facilities an architecture is next to useless. Without bug fixes and addition of new features to meet changing requirements it will not evolve and fall into disuse. Without occasional help from experts users will learn more slowly than they could.

Use case 3: Localisation and internationalisation

Goal summary To allow the use of the architecture in and for different languages.

Brief description Users of the architecture need to be able to have menus and at least some of the documentation in a familiar language, and they need to be able to build applications which process and display virtually any human language.

Use case 4: Software development good practice

Goal summary To promote good software engineering in LE development.

Brief description We can derive a number of general desiderata for SALEs on the basis that they are used for software development. In common with other software developers, SALE users need extensibility; interoperability; openness; explicit design documentation in appropriate modelling languages; graphical development environments; usage examples, or patterns.

Use case 5: Framework requirements

Goal summary To exploit the benefits of the framework.

Brief description Some general requirements for frameworks:

Orthogonality of elements: a user shouldn't have to learn everything in order to use one thing.

Availability of abstractions at different levels of complexity: a user should be able to do something basic in a simple fashion, but also be able to fiddle under the hood in a more complex scenario if necessary.

5.2. Components, PRs and LRs

5.2.1. Locating, loading and initialising components

Use case 6: Locate and load components

Goal summary To discover components at runtime, load and initialise them.

Brief description R&D developers create LR and PR components and reuse those created by others. Experimenters, students and teachers use components provided for them. Systems administrators install components. Applications developers embedd components in their systems. The set of components required in different cases is dynamic and loading should be dynamic as a consequence. The SALE should find all available components given minimal clues (perhaps a list of URLs), load them and initialise them ready for use.

5.2.2. Executing processing components

Use case 7: PR and LR Management

Goal summary To allow the building of systems from sets of components.

Brief description Developers need to be able to choose a subset of the available components and wire them together to form systems. These configurations should be shareable with other developers.

Use case 8: Distributed Processing

Goal summary To allow the construction of systems based on components residing on different host computers.

Brief description Components developed on one computer platform are seldom easy to move to other platforms. In order to reuse a diverse set of such components they must be made available over the net for distributed processing. Networks are often slow, however, so there must also be the capability to do all processing on one machine if the component set allows.

Use case 9: Parallel Processing

Goal summary To allow asynchronous execution of processing components.

Brief description Certain tasks can be carried out in parallel in some language processing systems. This implies that the execution of PRs should be multithreaded and means made available for parallel execution.

5.2.3. Representing information about components.

Use case 10: Component metadata

Goal summary To allow the association of structured data with LR and PR components.

Brief description Components are wired together with executive and task-specific code to form experimental systems or applications. Component metadata helps automate the wiring process, e.g. by describing the I/O contraints of the component. To use components they have to be found: metadata can be used to allow categorisation and description for browsing component sets.

5.2.4. Factoring out commonalities amongst components.

Use case 11: Component commonalities

Goal summary To factor out commonalities between related components.

Brief description Where there are families of components that share certain characteristics those commonalities should be modelled in the architecture. For example language analyser PRs characteristically take a document as input and add certain annotations to the document. Developers of analysers should be able to extend a part of the model which captures this and other characteristics.

5.2.5. Accessing data components

Use case 12: LR access

Goal summary To provide uniform, simple methods for accessing data components.

Brief description Just as the execution of PRs should be normalised by a SALE, so access to data components should be done in a uniform and efficient manner.

5.2.6. Managing collections of documents their formats

Use case 13: Corpora (Language Data LRs)

Goal summary To manage (possibly very large) collections of documents in an efficient manner.

Brief description Documents (texts and audiovisual materials) are grouped into collections which may have data associated with them. Operations which relate to documents should be generalisable to collections of documents.

Use case 14: Format-Independent Document Processing

Goal summary To allow SALE users to use documents of various formats without knowledge of those formats.

Brief description Documents can be processed independent of their formats. For example, an IE system can get to the text in an RTF[1] document or an HTML document without worrying about the structure of these formats. The structure is available for access where needed.

[1]Rich Text Format, Microsoft's Word document interchange format.

5.2.7. Representing information about text and speech

Use case 15: Annotations on Documents

Goal summary To support theory-neutral format-independent annotation of documents.

Brief description Many of the data structures produced and consumed by PR components are associated with text. Even NLG components can be viewed as producing data structures that relate to nascent texts that become progressively better specified, culminating in surface strings of words. See also interoperation use case (annotation import/export to/from SGML/XML).

5.2.8. Representing information about language

Use case 16: Data About Language LRs

Goal summary To support creation and maintenance of LRs that describe language.

Brief description Lexicons, grammars, ontologies, etc. etc. all require support tools for their development, for example for consistency checking, browsing and so on. (Note that this use case is potentially very large, and may fall outside of our scope.) In addition, developers of these types of resource use tools such as concordancers (e.g. KWIC) which should be provided by the development environment.

5.2.9. Indexing and retrieving information

Use case 17: Indices

Goal summary To cater for indexing and retrieval of diverse data structures.

Brief description The architecture includes data structures for annotating documents and for associating metadata with components. These data structures need efficient indices to make computation over large data sets tractable.

5.3. Method support

Use case 18: Common algorithms

Goal summary To provide a library of well-known algorithms over native data structures.

Brief description Although infrastructure should not in general stray into open research fields, where a particular algorithm is well-known it would be advantageous to provide a baseline implementation. For example, finite state transduction over annotation data structures, perhaps unification, ngram models and so on. (This use case is not under the annotation heading because it would be advantageous to generalise its application across other data structures and across text itself in some cases.)

Use case 19: Data comparison

Goal summary To provide simple methods for comparing data structures.

Brief description Machine learning methods, evaluation methods and introspective methods all need ways of comparing desired results on a particular language processing task with the results that a set of components has produced. In some cases this is a complex task (e.g. the comparison of MUC templates was found in some circumstances to be NP complete!), but in many cases a simple comparison measure based on identity is useful for a first-cut approximantion of success. This measure can be expressed as precision/recall where appropriate. (This use case is not under the annotation heading because it would be advantageous to generalise its application across other data structures and across text itself in some cases.)

5.4. Application issues

Use case 20: Persistence

Goal summary All data structures native to the architecture should be persistent.

Brief description The storage of data created automatically by components or manually by editing should be managed by the framework. This management should be transparent to a large degree, but must also be efficient and therefore should be amenable to tinkering where necessary. Access control may also be provided here.

Use case 21: Deployment

Goal summary To allow the use of the framework in diverse contexts.

Brief description The framework must be available in many context in order to allow the transfer of experimental and prototype systems from the development environment to external applications and parts of applications. Users must be able to use framework classes as a library, including classes of their own that are derived from the framework classes. They should also be able to build programs based on the framework by supplying their own executive code, and be able to access data resources from other contexts using standard database protocols.

5.5. Development issues

Use case 22: Interoperation and Embedding

Goal summary To enable data import from and export to other infrastructures and embedding of components in other environments.

Brief description Formats and formalisms for the expression of LRs come in many shapes and sizes. Some of these are dealt with by wrapping those formats in code that talks the language of the SALE framework. Other, widespread formats should be made more generally accessible via import/export filters. The prime case here is SGML/XML.

Certain common execution environments should be catered for, such as MS Office, OLE and Netscape Communicator.

Use case 23: Viewing and Editing

Goal summary To manipulate LE data structures.

Brief description SALEs are used to view and edit the data structures that LE systems process. This applies to both LRs and PRs.

Use case 24: Development UI

Goal summary To give access to all the framework and architectural services and support development of LE experiments and applications.

Brief description A large part of the story is components, which can be viewed, edited, stored, accessed from the framework API and so on. The final element is a UI for developers that wires all these together and gives top-level access to storage and component management, and execution of PRs.

6. References

Amtrup, J., 1999. Architecture of the Shiraz Machine Translation System. `http://crl.nmsu.edu/-shiraz/archi.html`.

Amtrup, J.W., 1995. ICE – INTARC Communication Environment User Guide and Reference Manual Version 1.4. Technical report, University of Hamburg.

Bird, S. and M. Liberman, 1998. Towards a Formal Framework for Linguistic Annotation. In *Proceedings of the ICLSP, Sydney*.

Bird, S. and M. Liberman, 1999a. A Formal Framework for Linguistic Annotation. Technical Report MS-CIS-99-01, Department of Computer and Information Science, University of Pennsylvania. `http://xxx.lanl.gov/-abs/cs.CL/9903003`.

Bird, S. and M. Liberman, 1999b. Annotation graphs as a framework for multidimensional linguistic data analysis. In *Towards Standards and Tools for Discourse Tagging, Proceedings of the Workshop. ACL-99*.

Bird, Steven, David Day, John Garofolo, John Henderson, Chris Laprun, and Mark Liberman, 2000. ATLAS: A flexible and extensible architecture for linguistic annotation. In *Proceedings of the Second International Conference on Language Resources and Evaluation*. Athens.

Boitet, C. and M. Seligman, 1994. The "Whiteboard" Architecture: A Way to Integrate Heterogeneous Components of NLP Systems. In *Proceedings of COLING '94*. Kyoto, Japan.

Booch, G., 1994. *Object-Oriented Analysis and Design 2nd Edtn.*. Benjamin/Cummings.

Bos, J., C.J. Rupp, B. Buschbeck-Wolf, and M. Dorna, 1998. Managing information at linguistic interfaces. In *Proceedings of the 36th ACL and the 17th COLING (ACL-COLING '98)*. Montreal.

Brew, C., D. McKelvie, R. Tobin, H. Thompson, and A. Mikheev, 1999. *The XML Library LT XML version 1.1 User documentation and reference guide*. Edinburgh: Language Technology Group. `http://www.ltg.ed.ac.uk/`.

Brugman, H., H.G. Russel, and P. Wittenburg, 1998. An infrastructure for collaboratively building and using multimedia corpora in the humaniora. In *Proceedings of the ED-MEDIA/ED-TELECOM Conference*. Freiburg.

Buckley, C., 1998. TIPSTER Advanced Query (DN2). TIPSTER programme working paper.

Busemann, S., 1999. Constraint-Based Techniques for Interfacing Software Modules. In *Proceedings of the AISB'99 Workshop on Reference Architectures and Data Standards for NLP*. Edinburgh, U.K.: The Society for the Study of Artificial Intelligence and Simulation of Behaviour.

Cahill, L., C. Doran, R. Evans, C. Mellish, D. Paiva, M. Reape, D. Scott, and N. Tipper, 1999a. Towards a Reference Architecture for Natural Language Generation Systems. Technical Report ITRI-99-14; HCRC/TR-102, University of Edinburgh and Information Technology Research Institute, Edinburgh and Brighton.

Cahill, L. and M. Reape, 1998. Component Tasks in Applied NLG Systems. RAGS deliverable, `http://www.tri.bbrighton.ac.uk/-projects/rags/`.

Cahill, L.J., C. Doran, R. Evans, D. Paiva, D. Scott, C. Mellish, and M. Reape, 1999b. Achieving Theory-Neutrality in Reference Architectures for NLP: To What Extent is it Possible/Desirable? In *Proceedings of the AISB'99 Workshop on Reference Architectures and Data Standards for NLP*. Edinburgh, U.K.: The Society for the Study of Artificial Intelligence and Simulation of Behaviour.

Cahoon, B. and K.S. McKinley, 1996. Performance Evaluation of a Distributed Architecture for Information Retrieval. In *Proceedings of SIGIR '96*. Zurich.

Cheong, T.L., A.W.L. Kwang, A. Gunawan, G.A. Loo, L.C. Qwun, and S.H. Leng, 1994. A Pragmatic Information Extraction Architecture for the Message Formatting Export (MFE)

System. In *Proceedings of the 2nd Singapore Conference on Intelligent Systems (SPICIS '94)*. Singapore.

Christ, O., 1994. A Modular and Flexible Architecture for an Integrated Corpus Query System. In *Proceedings of the 3rd Conference on Computational Lexicography and Text Research (COMPLEX '94)*. http://xxx.lanl.gov/-abs/cs.CL/9408005.

Christ, O., 1995. Linking WordNet to a Corpus Query System. In *Proceedings of the Conference on Linguistic Databases*. Groningen.

Clarkson, P. and R. Rosenfeld, 1997. Statistical Language Modeling using the SMU-Cambridge Toolkit. In *Proceedings of ESCA Eurospeech*. Greece.

Clements, P.C. and L.M. Northrop, 1996. Software Architecture: An Executive Overview. Technical Report CMU/SEI-96-TR-003, Software Engineering Institute, Carnegie Mellon University.

Cockburn, A., 1997. Structuring Use Cases with Goals. *Journal of Object-Oriented Programming*, Sept-Oct and Nov-Dec.

Connolly, D., 1997. *XML: Principles, Tools and Techniques*. Sebastopol, California: O'Reilly.

Cunningham, H., 1999. JAPE: a Java Annotation Patterns Engine. Research Memorandum CS – 99 – 06, Department of Computer Science, University of Sheffield.

Cunningham, H., R.G. Gaizauskas, K. Humphreys, and Y. Wilks, 1999. Experience with a Language Engineering Architecture: Three Years of GATE. In *Proceedings of the AISB'99 Workshop on Reference Architectures and Data Standards for NLP*. Edinburgh, U.K.: The Society for the Study of Artificial Intelligence and Simulation of Behaviour.

Cunningham, H., R.G. Gaizauskas, and Y. Wilks, 1995. A General Architecture for Text Engineering (GATE) – a new approach to Language Engineering R&D. Technical Report CS – 95 – 21, Department of Computer Science, University of Sheffield. http://xxx.lanl.gov/abs/cs.CL/9601009.

Cunningham, H., K. Humphreys, R. Gaizauskas, and Y. Wilks, 1996a. TIPSTER-Compatible Projects at Sheffield. In *Advances in Text Processing, TIPSTER Program Phase II*. DARPA, Morgan Kaufmann, California.

Cunningham, H., K. Humphreys, R. Gaizauskas, and Y. Wilks, 1997a. GATE – a TIPSTER-based General Architecture for Text Engineering. In *Proceedings of the TIPSTER Text Program (Phase III) 6 Month Workshop*. DARPA, Morgan Kaufmann, California.

Cunningham, H., K. Humphreys, R. Gaizauskas, and Y. Wilks, 1997b. Software Infrastructure for Natural Language Processing. In *Proceedings of the Fifth Conference on Applied Natural Language Processing (ANLP-97)*. http://xxx.lanl.gov/abs/cs.CL/9702005.

Cunningham, H., W. Peters, C. McCauley, K. Bontcheva, and Y. Wilks, 1998a. A Level Playing Field for Language Resource Evaluation. In *Workshop on Distributing and Accessing Lexical Resources at Conference on Language Resources Evaluation, Granada, Spain*.

Cunningham, H., M. Stevenson, and Y. Wilks, 1998b. Implementing a Sense Tagger within a General Architecture for Language Engineering. In *Proceedings of the Third Conference on New Methods in Language Engineering (NeMLaP-3)*. Sydney, Australia.

Cunningham, H., Y. Wilks, and R. Gaizauskas, 1996b. GATE – a General Architecture for Text Engineering. In *Proceedings of the 16th Conference on Computational Linguistics (COLING-96)*. Copenhagen.

Cunningham, H., Y. Wilks, and R. Gaizauskas, 1996c. Software Infrastructure for Language Engineering. In *Proceedings of the AISB Workshop on Language Engineering for Document Analysis and Recognition*. Brighton, U.K.

Cunningham, H., Y. Wilks, and R.J. Gaizauskas, 1996d. New Methods, Current Trends and Software Infrastructure for NLP. In *Proceedings of the Conference on New Methods in Natural Language Processing (NeMLaP-2)*. Bilkent University, Turkey. http://xxx.lanl.gov/abs/cs.CL/9607025.

Cunningham, Hamish, 2000. Software Architecture for Language Engineering. Forthcoming.

Cutting, D., J. Pedersen, and P-K. Halvorsen, 1991. An Object-Oriented Architecture for Text Retrieval. In *Proceedings of RIAO '91*. Barcelona.

Edmondson, W. and J. Iles, 1994a. A Non-linear Architecture for Speech and Natural Language Processing. In *Proceedings of International Conference on Spoken Language Processing (ICSLP '94)*, volume 1. Yokohama, Japan.

Edmondson, W. and J. Iles, 1994b. Pantome: An architecture for speech and natural language processing. Paper distributed at Dept. of CS Seminar, Sheffield University.

Eriksson, M., 1996. ALEP. http://www.sics.se/-humle/projects/svensk/platforms.html.

Estival, D., A. Lavelli, K. Netter, and F. Pianesi (eds.), 1997. *Computational Environments for Grammar Development and Linguistic Engineering*. Association for Computational Linguistics. Madrid, ACL-EACL'97.

Fikes, R. and A. Farquhar, 1999. Distributed Repositories of Higly Expressive Reusable Ontologies. *IEEE Intelligent Systems*, 14(2):73–79.

Fischer, D., W. Mohr, and L. Rostek, 1996. A Modular, Object-Oriented and Generic Approach for Building Terminology Maintenance Systems. In *TKE '96: Terminology and Knowledge Engineering*. Frankfurt.

Fowler, Martin and Kendall Scott, 1997. *UML Distilled*. Reading, Massachusetts: Addison-Welsey.

Fowler, Martin and Kendall Scott, 2000. *UML Distilled, Second Edition*. Reading, Massachusetts: Addison-Welsey.

Gaizauskas, R.G., H. Cunningham, Y. Wilks, P. Rodgers, and K. Humphreys, 1996. GATE – an Environment to Support Research and Development in Natural Language Engineering. In *Proceedings of the 8th IEEE International Conference on Tools with Artificial Intelligence (ICTAI-96)*. Toulouse, France.

Goldfarb, C.F. and P. Prescod, 1998. *The XML Handbook*. Upper Saddle River, NJ: Prentice Hall.

Goldfarb, Charles F., 1990. *The SGML Handbook*. Oxford University Press.

Goni, J.M., J.C. Gonzalez, and A. Moreno, 1997. ARIES: A lexical platform for engineering Spanish processing tools. *Journal of Natural Language Engineering*, 3(4):317–347.

Görz, G., M. Kessler, J. Spilker, and H. Weber, 1996. Research on Architectures for Integrated Speech/Language Systems in Verbmobil. In *Proceedings of COLING-96, Copenhagen.*.

Grishman, R., 1997. TIPSTER Architecture Design Document Version 2.3. Technical report, DARPA. http://www.itl.nist.gov/div894/894.02/-related_projects/tipster/.

Hayes-Roth, F., 1994. Architecture-Based Acquisition and Development of Software: Guidelines and Recommendations from the ARPA Domain-Specific Software Architecture (DSSA) Program. Technical report, Techknowledge Federal Systems. http://www.oswego.com/dssa/, visited 29th March 1999.

Hendler, J. and K. Stoffel, 1999. Back-End Technology for High-Performance Knowledge Representation Systems. *IEEE Intelligent Systems*, 14(3):63–69.

Hendry, D.G. and D.J. Harper, 1996. An Architecture for Implementing Extensible Information-Seeking Environments. In *Proceedings of SIGIR-96.*.

Henrich, A., 1996. Document Retrieval Facilities for Repository-Based System Development Environments. In *Proceedings of SIGIR '96*. Zurich.

Hobbs, J.R., 1993. The Generic Information Extraction System. In *Proceedings of the Fifth Message Understanding Conference (MUC-5)*. Morgan Kaufmann, California. `http://www.itl.nist.gov/div894/894.02/-related_projects/tipster/gen_ie.htm`.

Ibrahim, M.H. and F.A. Cummins, 1989. TARO: An Interactive, Object-Oriented Tool for Building Natural Language Systems. In *IEEE International Workshop on Tools for Artificial Intelligence*. Los Angeles.

Ide, N., 1998a. Corpus Encoding Standard: SGML Guidelines for Encoding Linguistic Corpora. In *Proceedings of the First International Language Resources and Evaluation Conference*. Granada, Spain.

Ide, N., 1998b. Encoding Linguistic Corpora. In *Proceedings of the Sixth Workshop on Very Large Corpora*. Montreal.

Ide, N. and G. Priest-Dorman, 1999. Corpus Encoding Standard. `http://www.cs.vassar.edu/CES/`.

Isard, A., D. McKelvie, and H.S. Thompson, 1998. Towards a Minimal Standard for Dialogue Transcripts: A New Sgml Architecture for the HCRC Map Task Corpus. In *Proceedings of the 5th International Conference on Spoken Language Processing (ICSLP '98)*. Sydney.

Jing, H. and K. McKeown, 1998. Combining Multiple, Large-Scale Resources in a Reusable Lexicon for Natural Language Generation. In *Proceedings of the 36th ACL and the 17th COLING (ACL-COLING '98)*. Montreal.

Johnson, J.A. and R.S. Rosenberg, 1995. A Data Management Strategy for Transportable Natural Language Interfaces. *International Journal of Intelligent Systems*, 10(9):771–808.

Koning, J.L., M.H. Stefanini, and Yves Deamzeau, 1995. DAI Interaction Protocols as Control Strategies in a Natural Language Processing System. In *Proceedings of IEEE Conference on Systems, Man and Cybernetics*.

Li, L., D.A. Dahl, L.M. Norton, M.C. Linebarger, and D. Chen, 1998. A Test Environment for Natural Language Understanding Systems. In *Proceedings of the 36th ACL and the 17th COLING (ACL-COLING '98)*. Montreal.

LREC-1, 1998. Conference on Language Resources Evaluation (LREC-1). Granada, Spain.

LuperFoy, S., D. Loehr, D. Duff, K. Miller, F. Reeder, and L. Harper, 1998. An Architecture for Dialogue Management, Context Tracking, and Pragmatic Adaptation in Spoken Dialogue Systems. In *Proceedings of the 36th ACL and the 17th COLING (ACL-COLING '98)*. Montreal.

Mason, O., 1998. The CUE Corpus Access Tool. In *Workshop on Distributing and Accessing Linguistic Resources*. Granada. `http://www.dcs.shef.ac.uk/~hamish/dalr/`.

McKelvie, D., C. Brew, and H. Thompson, 1997. Using SGML as a Basis for Data-Intensive NLP. In *Proceedings of the fifth Conference on Applied Natural Language Processing (ANLP-97)*. Washington, DC.

McKelvie, D., C. Brew, and H.S. Thompson, 1998. Using SGML as a Basis for Data-Intensive Natural Language Processing. *Computers and the Humanities*, 31(5):367–388.

Mikheev, A. and S. Finch, Washington, DC, 1997. A Workbench for Finding Structure in Text. In *Fifth Conference on Applied NLP (ANLP-97)*.

Nelson, T., 1997. Embedded Markup Considered Harmful. In D. Connolly (ed.), *XML: Principles, Tools and Techniques*. O'Reilly, pages 129–134.

Netter, K. and F. Pianesi, 1997. Preface. In *Proceedings of the Workshop on Computational Environments for Grammar Development and Linguistic Engineering*. Madrid.

Olson, M.R. and B.S. Lee, 1997. Object Databases for SGML Document Management. In *IEEE International Conference on Systems Sciences*.

Paggio, P., 1998. Validating the TEMAA LE evalutation methodology: a case study on Danish spelling checkers. *Journal of Natural Language Engineering*, 4(3):211–228.

Paiva, D.S., 1998. A Survey of Applied Natural Language Generation Systems. Technical Report ITRI-98-03, Information Technology Research Institute, Brighton.

Peters, W., H. Cunningham, C. McCauley, K. Bontcheva, and Y. Wilks, 1998. Uniform Language Resource Access and Distribution. In *Workshop on Distributing and Accessing Lexical Resources at Conference on Language Resources Evaluation, Granada, Spain*.

Poirier, H., 1999. The XeLDA Framework (presentation at Baslow workshop on Distributing and Accessing Linguistic Resources, Sheffield, 1999). `http://www.dcs.shef.ac.uk/~hamish/dalr/-baslow/xelda.pdf`.

Rao, R., H.D. Jellinek S.K. Card, J.D. Mackinlay, and G.G. Robertson, 1992. The Information Grid: a Framework for Information Retrieval and Retrieval-Centered Applications. In *Proceedings of the fifth annual ACM symposium on User interface software and technology (UIST '92)*. Monterey, CA.

Reiter, E., 1994. Has a Consensus NL Generation Architecture Appeared, and is it Psycholinguistically Plausible? In *Proceedings of the Seventh International Workshop on Natural Language Generation (INLGW-1994)*. `http://xxx.lanl.gov/abs/CS.cl/9411032`.

Reiter, E., 1999. Are Reference Architectures Standardisation Tools or Descripticve Aids? In *Proceedings of the AISB'99 Workshop on Reference Architectures and Data Standards for NLP*. Edinburgh, U.K.: The Society for the Study of Artificial Intelligence and Simulation of Behaviour.

Reiter, E. and R. Dale, 1999. Building Natural Language Generation Systems. *Journal of Natural Language Engineering*, Vol. 3 Part 1.

Reiter, E. and R. Dale, 2000. *Building Natural Language Generation Systems*. Cambridge, U.K.: Cambridge University Press.

Schutz, J., G. Thurmair, and R. Cencioni, 1991. An Architecture Sketch of Eurotra-II. In *MT Summit III*. Washington D.C.

Simkins, N. K., 1992. ALEP User Guide. CEC Luxemburg.

Simkins, N. K., 1994. An Open Architecture for Language Engineering. In *First CEC Language Engineering Convention, Paris*.

Sparck-Jones, K. and J. Galliers, 1996. *Evaluating Natural Language Processing Systems*. Springer.

Sperberg-McQueen, C.M. and L. Burnard, 1994. *Guidelines for Electronic Text Encoding and Interchange (TEI P3)*. ACH, ACL, ALLC. `http://etext.virginia.edu/-TEI.html`.

Stefanini, M.H. and Yves Deamzeau, 1995. TALISMAN: a multi-agent system for Natural Language Processing. In *Proceedings of IEEE Conference on Advances in Artificial Intelligence, 12th Brazilian Symposium on AI*.

Thurmair, G., 1996. WP 4.1 Task S6: Text Handling, Detailed Functional Specifications. Technical Report WP41-S6-V1.1, Sail Labs Gmbh., Munich. LE project LE1-2238 AVENTINUS internal report.

TIPSTER, 1995. The Generic Document Detection System. http://www.itl.nist.gov/div894/894.02/-related_projects/tipster/gen_ir.htm.

Tracz, W., 1995. Domain-Specific Software Architecture (DSSA) Frequently Asked Questions (FAQ). http://www.oswego.com/dssa/faq/faq.html, visited 29th March 1999.

University of Essex, 1999. Description of the W3-Corpora website. http://clwww.essex.ac.uk/w3c/.

von Hahn, W., 1994. The Architecture Problem in Natural Language Processing. *Prague Bulletin of Mathematical Linguistics*, 61:48–69.

Wolinski, F., F. Vichot, and O. Gremont, 1998. Producing NLP-based On-line Contentware. In *Natural Language and Industrial Applications*. Moncton, Canada. http://xxx.lanl.gov/abs/cs.CL/9809021.

Young, S., D. Kershaw, J. Odell, D. Ollason, V. Valtchev, and P. Woodland, 1999. *The HTK Book (Version 2.2)*. Cambridge, UK: Entropic Ltd. ftp://ftp.entropic.com/pub/htk/.

Yourdon, E., 1989. *Modern Structured Analysis*. Prentice Hall, New York.

Yourdon, E., 1996. *The Rise and Resurrection of the American Programmer*. Prentice Hall, New York.

Zajac, R., 1992. Towards Computer-Aided Linguistic Engineering. In *Proceedings of COLING '92*. Nantes, France.

Zajac, R., 1998a. Feature Structures, Unification and Finite-State Transducers. In *International Workshop on Finite State Methods in Natural Language Processing*. Ankara, Turkey.

Zajac, R., 1998b. Reuse and Integration of NLP Components in the Calypso Architecture. In *Workshop on Distributing and Accessing Linguistic Resources*. Granada. http://www.dcs.shef.ac.uk/~hamish/dalr/.