Computational Language Systems, Architectures

Hamish Cunningham and Kalina Bontcheva

Department of Computer Science, University of Sheffield, Regent Court, 211 Portobello Street, Sheffield S1 4DP, UK.
`hamish@dcs.shef.ac.uk` `kalina@dcs.shef.ac.uk`

Contents

Abstract

This article introduces the field of architecture, or infrastructure, for language processing scientific experimentation and language technology applications development. The article: presents a critical review of the various approaches that have been taken in the field; analyses eleven categories of previous work and uses these categories to organise the discussion; looks in detail at algorithmic and data resource infrastructure; concludes with some thoughts on the future of the area.

Keywords

Software Architecture for Language Engineering, Language Processing Infrastructure, Information Extraction, Natural Language Analysis, Human Language Technology, Natural Language Engineering.

ELL2 Cross-References

"Natural Language Analysis". "Text Retrieval Conference and Message Understanding Conference". "Natural Language Processing, System Evaluation". "Human Language Technology". "Language Processing, Statistical Methods"

Acknowledgements

Thanks to Valentin Tablan and Diana Maynard for important contributions to the area covered in this article. The author was partly supported by EPSRC grant GR/N15764/01 (AKT) and by EU grants SEKT, PrestoSpace and KnowledgeWeb.

1 Introduction: Software Architecture

Every building, and every computer program, has an *architecture*: structural and organisational principles that underpin its design and construction. The garden shed once built by one of the authors had an ad hoc architecture, extracted (somewhat painfully) from the imagination during a slow and non-deterministic process that, luckily, resulted in a structure which keeps the rain on the outside and the mower on the inside (at least for the time being). As well as being *ad hoc* (i.e. not informed by analysis of similar practice or relevant science or engineering) this architecture is *implicit*: no explicit design was made, and no records or documentation kept of the construction process.

The pyramid in the courtyard of the Louvre, by contrast, was constructed in a process involving *explicit design* performed by qualified engineers with a wealth of theoretical and practical knowledge of the properties of materials, the relative merits and strengths of different construction techniques, *et cetera.*

So it is with software: sometimes it is thrown together by enthusiastic amateurs; sometimes it is *architected*, built to last, and intended to be ‘not something you finish, but something you start’ (to paraphrase Brand (1994)).

A number of researchers argued in the early and middle 1990s that the field of computational infrastructure or architecture for human language computation merited an increase in attention. The reasoning was that the increasingly large-scale and technologically significant nature of language processing science was placing increasing burdens of an engineering nature on research and development workers seeking robust and practical methods (as was the increasingly collaborative nature of research in this field, which puts a large premium on software integration and interoperation). [1]). Over the intervening period a number of significant systems and practices have been developed in what we may call Software Architecture for Language Engineering (SALE).

We may define Language Engineering (LE) as the production of software systems that involve processing human language with *quantifiable* accuracy and *predictable* development resources Cunningham (1999). LE is related to but distinct from the fields of Computational Linguistics, Natural Language Processing and Artificial Intelligence, with its own priorities

[1] Software lifecycle in collaborative research – how not to do it:

- **Project Proposal**: ‘We love each other. We can work so well together. We can hold workshops on Greek Islands together. We will solve all the problems of the field that our predecessors were too stupid to.’
- **Analysis and Design**: Stop work entirely, for a period of reflection and recuperation following the stress of attending the kick-off meeting.
- **Implementation**: Each developer partner tries to convince the others that program X that they just happen to have lying around on a dusty disk-drive meets the project objectives exactly and should form the centrepiece of the demonstrator.
- **Integration and Testing**: The lead partner gets desperate and decides to hard-code the results for a small set of examples into the demonstrator, and have a fail-safe crash facility for unknown input (‘well, you know, it’s still a prototype...’).
- **Evaluation**: Everyone says how nice it is, how it solves all sorts of terribly hard problems, and how if we had another grant we could go on to transform information processing the world over (or at least the business travel industry).

and concerns. Chief among these are: dealing with large-scale tasks of practical utility; measuring progress quantitatively relative to performance on examples of such tasks; a growing realisation of the importance of software engineering in general, and reusability, robustness, efficiency and productivity in particular. As argued in Cunningham and Scott (2004); Maynard et al. (2002) software architectures can contribute significantly towards achieving these goals.

This article gives a critical review of the various approaches that have been taken to the problem of Software Architecture for Language Engineering (SALE). The prime criterion for consideration in this article is being *infrastructural*, i.e. we consider work that is intended to support Language Engineering (LE) R&D in some way that extends beyond the boundaries of a single time-limited project. Section 2 expands the definition of SALE.

We cover eleven categories of work, which ranges over a wide area. In order to provide an organising principle for the discussion we begin in section 3 by extrapolating a set of architectural issues that represents the union of those addressed by the various researchers cited. This has the advantage of making it easier to see how certain problems have been addressed and the disadvantage that multipurpose infrastructures appear in several categories.

Section 4 discusses infrastructures aimed at algorithmic resources, including the issues of component integration and execution. Section 5 turns to data resources infrastructure, including the issues of access and the representation of information about text and speech. Section 6 concludes by providing a discussion on future directions for work on SALE.

2 Software Architectures for Language Engineering

The problem addressed by the systems reviewed here 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 targeted 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) "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 conform to architectural principles and visualise the effect of various design choices and can provide code libraries tailored to the architecture.

The most significant issues addressed by SALE systems include:

(1) Clean separation of low-level tasks such as data storage, data visualisation, location and loading of components, and execution of processes from the data structures and algorithms that actually process human language.
(2) Reducing integration overheads by providing standard mechanisms for components to communicate data about language, and using open standards such as Java and XML as the underlying platform.
(3) Providing a baseline set of language processing components that can be extended and/or replaced by users as required.
(4) Providing a development environment or at least a set of tools to support users with modification and implementation of language processing components and applications.
(5) Automating measurement of performance of language processing components.

Here we will focus on the first two sets of issues, since they are issues that arise in every single NLP system or application, and are prime areas where SALE can contribute. For a discussion of other requirements see Cunningham (2000).

3 Categories of Work on SALE

Like other software, Language Engineering (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 (see e.g., Booch (1994)).

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, thesauri 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.

We will use this distinction to categorise work on SALE. Section 4 will survey infrastructural work on processing resources, while section 5 reviews the much more substantial body of work on language resources. These sections are additionally segmented into categories as follows:

- **Processing Resources**
  - Locating, loading and initialising components from local and non-local machines.
  - Executing processing components, serially or in parallel.
  - Representing information about components.
  - Factoring out commonalities amongst components.
- **Language Resources, corpora and annotation**
  - Accessing data components.
  - Managing collections of documents (including recordings) and their formats.
  - Representing information about text and speech.
  - Representing information about language.
  - Indexing and retrieving information.

4 Processing Resources

It is commonly the case that a language processing system is composed of a number of discrete steps. For example, a translation application must first analyse the source text in order to arrive at some representation of meaning before it can begin deciding upon target language structures that parallel that meaning. A typical language analysis process will implement stages such as text structure analysis, tokenisation, morphological analysis, syntactic parsing, and semantic analysis[2]. Each of these stages is represented by components

[2] The exact breakdown varies widely and is to some extend dependent on method; some statistical work early in the second wave of the application of these types of method completely ignored the conventional language analysis steps in favour of a technique based on a memory of parallel texts Brown et al. (1990). Later work has tended to accept the advantages of some of these stages,

that perform processes on text (and use components containing data about language such as lexicons and grammars). In other words, the analysis steps are realised as a set of Processing Resources (PRs). Several architectural questions arise in this context:

(1) Is the execution of the PRs best done serially or in parallel?
(2) How should PRs be represented such that their discovery on a network and loading into an executive process is transparent to the developer of their linguistic functions?
(3) How should distribution across different machines be handled?
(4) What information should be stored about components and how should this be represented?
(5) How can commonalities between component sets be exploited?
(6) How should the components communicate information between each other? (This question can also be stated as: how should information about text and speech be represented?)

This section reviews work that addresses questions 1 – 5. The issue of representing information about language is addressed in section 5.

4.1 Locating and Loading

There are several reasons why PR components should be separate from the controlling application that executes them:

- There will often be a many-to-one relation between applications and PRs. Any application using language analysis technology needs a tokeniser component, for example.
- A PR may have been developed for one computing platform, e.g. UNIX, but the application wishing to use it may operate on another, e.g. Windows.
- The processing regime of the application may require linear or asynchronous execution; this choice should be isolated from the component structures as far as possible in order to promote generality and encourage reuse.
- PR developers should not be forced to deal with application-level software engineering issues such as how to manage installation, distribution over networks, exception handling and so on.
- Explicit modelling of components allows exploitation of modern component infrastructures such as Java Beans or Active X.

Accordingly, many papers on infrastructural software for LE separate components from the control executive,[3] e.g., Boitet and Seligman (1994); Edmondson and Iles (1994); Koning et al. (1995); Wolinski et al. (1998); Poirier (1999); Zajac (1998b); Lavelli et al. (2002);

however, though they may be moved into an off-line corpus annotation process such as Marcus et al. (1993).

[3] The term 'executive' is used here in the sense of a software entity which executes, or runs, other entities.

Cunningham et al. (2002a). The question then is how components become known to control processes or applications, and how they are loaded and initialised. (A related question is what data should be stored with components to facilitate their use by an executive – see section 4.3 below.) Much work ignores this issue; the rest of this section covers those SALE systems for which the data is available.

The TIPSTER architecture Grishman (1997) recognised the existence of the locating and loading problems, but didn't provide a full solution to the problem. The architecture document includes a place-holder for such a solution (in the form of a 'register annotator' API[4] call, which an implementation could use to provide component loading), but the semantics of the call were never specified.

The TalLab architecture "is embedded in the operating system" Wolinski et al. (1998) which allows them to "reuse directly a huge, efficient and reliable amount of code". The precise practicalities of this choice are unclear, but it seems that components are stored in particular types of directory structure, which are presumably known to the application at startup time.

ICE, the Intarc Communication Environment Amtrup (1995), is an 'environment for the development of distributed AI systems', and part of the Verbmobil real-time speech-to-speech translation project Kay et al. (1994). ICE provides distribution based around PVM (Parallel Virtual Machine) and a communication layer based on channels. ICE is not specific to LE because the communication channels do not use data structures specific to NLP needs, and because document handling issues are left to the individual modules. ICE's answer to the locating and loading problem is the Intarc License Server, which is a kind of naming service, or registry, that stores addressing information for components. Components must themselves register with the server by making an API call (`Ice_Attach`). The components must therefore a) link to the ICE libraries and b) know the location of the license server (as must applications using ICE services).

Following from the ICE work Herzog et al. (2004) present the latest in three generations of architecture to arise from the Verbmobil and Smartkom projects, in the shape of the Multiplatform system. This architecture supports multiple distributed components from diverse platforms and implementation languages running asynchonously and communicating via a message passing substrate.

Corelli Zajac (1997) and its successor Calypso Zajac (1998b) are also distributed systems that cater for asynchronous execution. The initial Corelli system implemented much of the CORBA standard The Object Management Group (1992), and component discovery used a naming and directory service. All communication and distribution was mediated by an Object Request Broker (ORB). Components ran as servers and implemented a small API to allow their use by an executive or application process. In the later Calypso incarnation, CORBA was replaced with simpler mechanisms due to efficiency problems (for a usage example see Amtrup (1999)). In Calypso components are stored in a centralised repository, sidestepping the discovery problem. Loading is catered for by requiring components to im-

[4] Application Programmers' Interface.

plement a common interface.

Another distributed architecture based on CORBA is SiSSA Lavelli et al. (2002). The architecture comprises processors (PRs in our terms), servers for their execution, data containers (LRs) and a manager component (called SiSSA Manager) which establishes and removes connections between the processors, according to a user-designed data flow. SiSSA uses a *Processor Repository* to keep information about processors registered with the architecture.

Carreras and Padró (2002) report a distributed architecture specifically for language analysers.

GATE version 1 Cunningham et al. (1997) was a single-process, serial execution system. Components must reside in the same file system as the executive; location was performed by searching a path stored in an environment variable. Loading was performed in three ways, depending on the type of component and which of the GATE APIs it uses.

GATE version 2 Cunningham et al. (2002a,b) supports remote components; location is performed by providing one or more component repositories (called CREOLE[5] repositories), which contain XML definitions of each resource and the types of its parameters (e.g., whether it works with documents or corpora). The user can then instantiate a component by selecting it from the list of available components and choosing its load time parameters. GATE makes distinction between load time and runtime parameters – the former are essential for the working of the module, e.g., a grammar, and need to be provided at load time, while the latter can change from one execution to the next, e.g. a document to be analysed. Components can also be re-initialised, which enables users to edit their load-time data (e.g., grammars) within the graphical environment and then re-load the component to reflect the changes. GATE also supports editing of remote language resources and execution of remote components using Remote Method Invocation (RMI), i.e., provides facilities for building client-server applications.

4.2 Execution

It seems unlikely that people process language by means of a set of linear steps involving morphology, syntax and so on. More probably we deploy our cognitive faculties in a parallel fashion (hence, for example, the term ‘parallel distributed processing’ in neural modelling work McClelland and Rumelhart (1986)). These kinds of ideas have motivated work on non-linear component execution in NLP; von Hahn (1994) gives an overview of a number of approaches; a significant early contribution was the Hearsay speech understanding system Erman et al. (1980).

Examples of asynchronous infrastructural systems include Kasuga Boitet and Seligman (1994), Pantome Edmondson and Iles (1994), Talisman Koning et al. (1995), Verbmobil

[5] Collection of REusable Objects for Language Engineering.

Görz et al. (1996), TalLab Wolinski et al. (1998), Xelda Poirier (1999), Corelli Zajac (1997), Calypso Zajac (1998b), SiSSA Lavelli et al. (2002), Distributed Inquery Cahoon and McKinley (1996), and the Galaxy Communicator Software Infrastructure (GCSI) MITRE (2002). Motivations include that mentioned in the previous paragraph, and also the desire for feedback loops in ambiguity resolution (see Koning et al. (1995)).

In the Inquery and Verbmobil cases an additional motivation is efficiency. ICE, the Verbmobil infrastructure, addressed two problems: distributed processing and incremental interpretation. Distribution is intended to contribute to processing speed in what is a very compute-intensive application area (speech-to-speech translation). Incremental interpretation is both for speed and to facilitate feedback of results from downstream modules to upstream ones (e.g. to inform the selection of word interpretations from phone lattices using part-of-speech information). ICE's PVM-based architecture provides for distributed asynchronous execution.

GCSI is an open source architecture for constructing dialogue systems. This infrastructure concentrates on distributed processing, hooking together sets of servers and clients that collaborate to hold dialogues with human interlocutors. Data gets passed between these components as attribute/value sets, or 'frames', the structuring and semantics of which must be agreed upon on a case-by-case basis. Communication between modules is achieved using a hub. This architectural style tends to treat components as black boxes which are developed using other tool sets. To solve this problem, other support environments can be used to produce a GCSI server components, using GCSI as a communication substrate to integrate with other components.

The model currently adopted in GATE is that each PR may run in its own thread if asynchronous processing is required (by default PRs will be executed serially in a single thread). The set of LRs being manipulated by a group of multi-threaded PRs must be synchronised (i.e. all their methods must have locks associated with whichever thread is calling them at a particular point). Synchronisation of LRs is performed in a manner similar to the Java collections framework. This arrangement allows the PRs to share data safely. Responsibility for the semantics of the interleaving of data access (who has to write what in what sequence in order for the system to succeed) is a matter for the user, however.

4.3 Metadata

A distinction may be made between the data that language processing components use (or Language Resources) and data that is associated with components for descriptive and other reasons. The latter is sometimes referred to as *metadata* to differentiate it from the former. In a similar fashion, the content of the Web is largely expressed in HTML; data that *describes* Web resources, e.g. 'this HTML page is a library catalogue', is also called metadata. Relevant standards in this area include RDF, the Resource Description Framework Lassila and Swick (1999); Berners-Lee et al. (1999).

There are several reasons why metadata should be part of a component infrastructure, including:

- to facilitate the interfacing and configuration of components;
- to encode version, author and availability data;
- to encode purpose data and allow browsing of large component sets.

When components are reused across more than one application or research project it will often be the case that their input/output (I/O) characteristics have not been designed alongside the other components forming the language processing capability of the application. For example, one part-of-speech tagger may require tokens as input in a one-per-line encoding. Another may require SGML[6] input. In order to reuse the tagger with a tokeniser that produces some different flavour of output, that output must be transformed to suit the tagger's expectations. In cases where there is an isomorphism between the available output and the required input a straightforward syntactic mapping of representations will be possible. In cases where there is a semantic mismatch, additional processing will be necessary.

Busemann (1999) addresses component interfacing and describes a method for using feature structure matrices to encode structural transformations on component I/O data structures. These transformations are essentially re-ordering of the data structures around pre-existing unit boundaries; therefore the technique assumes isomorphism between the representations concerned. The technique also allows for type checking of the output data during restructuring.

TIPSTER Grishman (1997), GATE Cunningham (2002) and Calypso Zajac (1998b) deal with interfacing in two ways. First, component interfaces share a common data structure (e.g. corpora of annotated documents), thus ensuring that the syntactic properties of the interface are compatible. Component wrappers are used to interface to other representations as necessary (so, for example, a Brill tagger Brill (1992) wrapper writes out token annotations in the required one-per-line format, then reads in the tags and writes them back to the document as annotations). Second, where there is semantic incompatibility between the output of one component and the input of another a dedicated transduction component can be written to act as an intermediary between the two.

In Verbmobil a component interface language is used Bos et al. (1998), which constrains the I/O profiles of the various modules. This language is a Prolog term which encodes logical semantic information in a flat list structure. The principle is similar to that used in TIPSTER-based systems, but the applicability is somewhat restricted by the specific nature of the data structure.

Provision of descriptive metadata has been addressed by the Natural Language Software Registry (NLSR) DFKI (1999) and by the EUDICO distributed corpora project Brugman et al. (1998a,b). In each case Web-compatible data (HTML and XML respectively) are associated with components. The NLSR is purely a browsable description; the EUDICO

[6] The Standard Generalised Markup Language Goldfarb (1990).

work links the metadata with the resources themselves, allowing the launching of appropriate tools to examine them with[7]. GATE 2 Cunningham et al. (2002b) uses XML for describing the metadata associated with processing resources in its CREOLE repositories (see section 4.1). This metadata is used for component loading and also for launching the corresponding visualisation and editing tools.

In addition to the issue of I/O transformation, in certain cases it may be desirable to be able to identify automatically which components are plug-compatible with which other ones, in order to identify possible execution paths through the component set.

GATE 1 Cunningham et al. (1997) addresses automatic identification of execution paths by associating a configuration file with each processing component that details the input (pre-conditions) and output (post-conditions) in terms of TIPSTER annotation and attribute types (see section 5.3.2 below). This information is then used to auto-generate an execution graph for the component set.

4.4 Commonalities

To conclude our survey of infrastructural work related to processing, this section looks at the exploitation of commonalities between components. For example, both parsers and taggers have the characteristics of language analysers. One of the key motivating factors for SALE is to break the 'software waste cycle' Veronis and Ide (1996) and promote reuse of components. Various researchers have approached this issue by identifying typical component sets for particular tasks, e.g., Hobbs (1993); TIPSTER (1995); Reiter and Dale (2000). Some work goes on to provide implementations of common components Cheong et al. (1994); Ibrahim and Cummins (1989). The rest of this section describes these approaches.

Reiter and Dale have reviewed and categorised Natural Language Generation (NLG) components and systems in some detail. Reiter (1994) argues that a consensus component breakdown has emerged in NLG (and that there is some psychological plausibility for this architecture); the classification is extended in Reiter and Dale (2000). They also discuss common data structures in NLG (as does the RAGS project – see section 5.4 below), and appropriate methods for the design and development of NLG systems. Reiter (1999) argues that the usefulness of this kind of architectural description is to 'make it easier to describe functionalities and data structures' and thus facilitate research by creating a common vocabulary amongst researchers. He states that this is a more limited but more realistic goal than supporting integration of diverse NLG components in an actual software system. The term he uses for this kind of descriptive work is a 'reference architecture', which is also the subject of the workshop at which the paper was presented Mellish and Scott (1999).

The TIPSTER research programme developed descriptive or reference architectures for Information Extraction and for Information Retrieval. Hobbs (1993) describes a typical module

[7] Note that EUDICO has only dealt with Language Resource components at the time of writing.

set for an IE system. The architecture comprises 10 components, dealing with tasks like preprocessing, parsing, semantic interpretation, and lexical disambiguation; for a description of the full set see Gaizauskas and Wilks (1998). For IR TIPSTER (1995) describes two functions, search and routing, each with a typical component set (some of which are PRs and some LRs.)

An architecture for spoken dialogue systems is presented in LuperFoy et al. (1998), which divides the task into dialogue management, context tracking and pragmatic adaptation. This in turn leads to an architecture in which various components (realised as agents) collaborate in the dialogue. Some example components are: speech recognition, language interpretation, language generation, and speech synthesis. In addition a dialogue manager component provides high-level control and routing of information between components.

The preceding discussion illustrates that there is considerable overlap between component sets developed for various purposes. A SALE that facilitated multipurpose components would cut down on the waste involved in continual re-implementation of similar components in different contexts. The component model given in Cunningham (2000) is made available in the GATE framework Cunningham et al. (2002b). This model is based on inheritance: a parser is a type of language analyser which is a type of processing resource. The language engineer can choose, therefore, between implementing a more specific interface and adhering to the choices made by the GATE developers for that type, or implementing a more general interface and making their own choices about the specifics of their particular resource.

In several cases, work on identifying component commonalities has led to the development of toolkits that aim to implement common tasks in a reusable manner. For example: TARO Ibrahim and Cummins (1989) is an OO syntactic analyser toolkit based on a specification language. A toolkit for building IE systems and exemplified in the MFE IE system is presented in Cheong et al. (1994).

5 Language Resources

Recall from section 3 that Language Resources are data components such as lexicons, corpora and language models. They are the raw materials of language engineering. This section covers six issues relating to infrastructure for LRs:

(1) Computational access (local and non-local).
(2) Managing document formats and document collections (corpora), including multilingual resources.
(3) Representing information about corpora (language data, or performance modelling).
(4) Representing information about language (data about language, or competence modelling).
(5) Indexing and retrieval of language-related information.

Note also that the advantages of a component-based model presented (in relation to PRs) in section 4.1 also apply to LRs.

5.1 Programmatic Access

LRs are of worth only inasmuch as they contribute to the development and operation of PRs and the language processing research prototypes, experiments and applications that are built from them. A key issue in the use of LRs for language processing purposes is that of computational access. Suppose that a developer is writing a program to generate descriptions of museum catalogue items; they may have a requirement for synonyms, for example, in order to lessen repetition. A number of sources for synonyms are available, e.g. WordNet Miller (Ed.), or Roget's *Thesaurus*. In order to reuse these sources the developer needs to access the data in these LRs from their program.

Although the reuse of LRs has exceeded that of PRs Cunningham et al. (1994) in general, there are still two barriers to LR access and hence LR reuse:

(1) each resource has its own representation syntax and corresponding programmatic access mode (e.g. SQL for Celex, C or Prolog for WordNet);
(2) resources must generally be installed locally to be usable, and how this is done depends on what operating systems are available, what support software is required, etc., which varies from site to site.

A consequence of (1) is that although resources of the same type usually have some structure in common (for example, at one of the most general levels of description, lexicons are organised around words), this commonality cannot be exploited when it comes to using a new resource. In each case the user has to adapt to a new data structure; this adaptation is a significant overhead. Work which seeks to investigate or exploit commonalities between resources has first to build a layer of access routines on top of each resource. So, for example, if we wished to do task-based evaluation of lexicons by measuring the relative performance of an IE system with different instantiations of lexical resource, we would typically have to write code to translate several different resources into SQL or some other common format. Similarly, work such as Jing and McKeown (1998) on merging large scale lexical resources (including WordNet and Comlex) for NLG has to deal with this problem.

A consequence of (2) is that users may have to adjust their compute environments to suit resources tailored to particular platforms. Also, there is no way to 'try before you buy': no way to examine an LR for its suitability for one's needs before licensing it *in toto*. Correspondingly there is no way for a resource provider to give limited access to their products for advertising purposes, or gain revenue through piecemeal supply of sections of a resource.

There have been two principal responses to problem (1): standardisation and abstraction.

The standardisation solution seeks to impose uniformity by specifying formats and structures for LRs. So, for example, the EAGLES working groups have defined a number of standards for lexicons, corpora and so on EAGLES (1999). More recently Ide and Romary (2004) report the creation of a framework for linguistic annotations as part of the work of ISO standardisation Technical Committee 37, Sub-Committee 4, whose objective

> . . . is to prepare various standards by specifying principles and methods for creating, coding, processing and managing language resources, such as written corpora, lexical corpora, speech corpora, dictionary compiling and classification schemes. These standards will also cover the information produced by natural language processing components in these various domains.[8]

The work reported here is from Working Group 1 of the committee, which has developed a linguistic annotation framework based on the XML, RDF(S), and OWL.

While standardisation would undoubtedly solve the representation problem, there remains the question of existing LRs (and of competing standards). Peters et al. (1998); Cunningham et al. (1998) describe experiments with an abstraction approach based on a common Object-Oriented model for LRs that encapsulates the union of the linguistic information contained in a range of resources, and encompasses as many object hierarchies as there are resources. At the top of the resource hierarchies are very general abstractions; at the leaves are data items specific to individual resources. Programmatic access is available at all levels, allowing the developer to select an appropriate level of commonality for each application. Generalisations are made over different object types in the resources, and the object hierarchies are linked at whatever levels of description are appropriate. No single view of the data is imposed on the user, who may choose to stay with the 'original' representation of a particular resource, or to access a model of the commonalities between several resources, or a combination of both.

Problem (2), non-local access, has also attracted two types of response, which can be broadly categorised as: Web browsing; distributed databases.

A number of sites now provide querying facilities from HTML pages, including:

- the Linguistic Data Consortium at `http://www.ldc.upenn.edu/`;
- the British National Corpus server at `http://info.ox.ac.uk/bnc/`.

So, for example, all occurrences of a particular word in a particular corpus may be found via a Web browser. This is a convenient way to access LRs for manual investigative purposes, but is not suited to (or intended for) use by programs for their access purposes.

Moving beyond browsing, several papers report work on programmatic access using distributed databases. Fikes and Farquhar (1999) show how ontologies may be distributed; Brugman et al. (1998a,b) describe the EUDICO distributed corpus access system; Peters et al. (1998); Cunningham et al. (1998) propose a system similar to EUDICO, generalised to other types of LR. Some new directions in sharing language resources are discussed in section 6.

[8] `http://www.tc37sc4.org/`

Other issues in the area of access to LRs include that of efficient indexing and search of corpora (see section 5.5), and that of annotation of corpora (see section 5.3). The issue of how to access SGML documents in an efficient manner is discussed in Olson and Lee (1997), who investigated the use of Object-Oriented databases for storing and retrieving SGML documents. Their conclusions were essentially negative due to the slowness of the databases used. Hendler and Stoffel (1999) discuss how ontologies may be stored and processed efficiently using relational databases, and here the results are more positive.

5.2 Documents, Formats and Corpora

Documents play a central role in LE. They are the subject of analysis for technologies such as IE; they are both analysed and generated in technologies such as MT. In addition a large amount of work uses annotated documents as training data for machine learning of numerical models. Previous work on LE infrastructure has developed models for documents and corpora, provided abstraction layers for document formats, and investigated efficient storage of documents in particular formats.

Documents may contain text, audio, video or a mixture of these (the latter are referred to as multimedia documents). The underlying data will frequently be accompanied by formating information (delineating titles, paragraphs, areas of bold text, etc.) and, in the LE context, annotation (storing linguistic data such as gesture tags, part-of-speech tags or syntax trees). Both formatting and annotation come in a wide variety of flavours. Formats include proprietary binary data such as MS Word's `.doc` or Excel's `.xls`, semi-open semi-readable formats such as Rich Text Format (Word's exchange format), and non-proprietary standardised formats such as HTML, XML, or GIF (Graphics Interchange Format).

The Text Encoding Initiative (TEI) Sperberg-McQueen and Burnard (1994, 2002) and the Corpus Encoding Standard (CES Ide (1998) and XCES Ide et al. (2000)) are models of documents and corpora that aim to standardise the representation of structural and linguistic data for textual documents. The general approach is to represent *all* information about document structure, formatting and linguistic annotation using SGML/XML.

The issue of document formats has been addressed by several TIPSTER-based systems, including GATE and Calypso, and by the HTK speech recognition toolkit Young et al. (1999). In the latter case the approach is to provide API calls that deal with documents in various known formats (e.g. WAV [9] , MPEG) independent of those formats. So, for example, a speech recogniser can access the raw audio from these documents without knowing anything about the representation format.

The TIPSTER systems deal with formats by means of input filters that contain knowledge about the format encoding and use that knowledge to unpack format information into annotations. TIPSTER also supplies a model of corpora and data associated with both corpus

[9] Windows Audio format.

and documents Grishman (1997). Note that the two approaches are not mutually exclusive: Ogden (1999) has defined a mapping between TEI/CES and TIPSTER annotations.

Another important issue that needs to be dealt with in infrastructures supporting language resources in multiple languages is the problem of editing and displaying multilingual information. It is often thought that the character sets problem is solved by use of the Unicode standard. This standard is an important advance, but in practice the ability to process text in a large number of the world's languages is still limited by:

- incomplete support for Unicode in operating systems and applications software;
- languages missing from the standard;
- difficulties in converting non-Unicode character encodings to Unicode.

In order to deal with all these issues, including displaying and editing of Unicode documents, GATE provides a Unicode Kit and a specialised editor Tablan et al. (2002). In addition, all processing resources and visualisation components are Unicode-compliant.

5.3 Annotation

One of the key issues for much work in this area is how to represent information about text and speech. (This kind of information is sometimes called *language data*, distinguishing it from *data about language* in the form of lexicons, grammars, etc.)

Two broad approaches to annotation have been taken: to use markup (e.g., SGML/XML); to use annotation data structures with references or pointers to the original (e.g., TIPSTER, ATLAS).

Interestingly, the differences between the two kinds of approaches have become less pronounced in recent work. SGML used to generally involve embedding markup in the text; TIPSTER (and related systems) use a referential scheme where the text remains unchanged and annotation refers back to it by character offsets. The embedding approach has a number of problems Nelson (1997); Cunningham et al. (1997), including the difficulty of extending the model to cope with multimedia data Bird and Liberman (1999a). Partly in response to these difficulties, and as part of the rebirth of SGML as XML Goldfarb and Prescod (1998), the 'ML' community has adopted a referential scheme itself, which is now known as 'standoff markup'. The data models of the various systems are now much closer than they were before XML existed, and the potential for interoperation between referential systems like GATE and XML-based architectures is greater as a result. GATE exploits this potential by providing input from and output to XML in most parts of the data model Cunningham et al. (2002a,b).

5.3.1 Markup-based architectures

Language data can be represented by embedding annotation in the document itself (at least in the case of text documents; users of embedding typically transcribe speech documents before markup, or use 'stand-off markup' – see below). The principal examples of embedded markup for language data use the Standard Generalized Markup Language (SGML Goldfarb (1990); XML is discussed below). SGML is a *meta-language*, a language used to create other languages. The syntax of SGML is therefore abstract, with each document filling in this syntax to obtain a concrete syntax and a particular markup language for that document. In practice certain conventions are so widespread as to be *de facto* characteristics of SGML itself. For example, annotation is generally delimited by `<TAG>` and `</TAG>` pairs, often with some attributes associated such as `<TAG ATTRIBUTE=value>`. The legitimate tags (or *elements*) and their attributes and values must be defined for each class of document, using a Document Type Definition (DTD). It does *not* specify what the markup means; the DTD is the grammar which defines how the elements may be legally combined and in what order in a particular class of text; see Goldfarb (1990). A good example of SGML used for corpus annotation is the British National Corpus (BNC) Burnard (1995).

The HyperText Markup Language (HTML) is an application of SGML, and is specified by its own DTD. A difference with ordinary SGML is that the DTD is often cached with software such as Web browsers, rather than being a separate file associated with the documents that instantiate it. In practice Web browsers have been lenient in enforcing conformance to the HTML DTD, and this has led to diversity amongst Web pages, meaning that HTML DTDs now represent an idealised specification of the language which often differs from its usage in reality.

Partly in response to this problem the eXtensible Markup Language, XML Goldfarb and Prescod (1998) has been developed. SGML is a complex language: DTDs are difficult to write, and full SGML is difficult to parse. XML made the DTD optional, and disallowed certain features of SGML such as markup minimisation. For example, the American National Corpus (ANC) Macleod et al. (2002) uses XML and XCES Ide et al. (2000) to encode linguistic annotations.

One of the problems in the SGML/XML world is that of computational access to and manipulation of markup information. Addressing this problem, the Language Technology group at the University of Edinburgh developed an architecture and framework based on SGML called the LT Normalised SGML Library (LT NSL) McKelvie et al. (1998). This in turn led to the development of LT XML Brew et al. (1999) tracking the introduction of the XML standard.

Tools in an LT NSL system communicate via interfaces specified as SGML DTDs (essentially tag set descriptions), using character streams on pipes: a pipe-and-filter arrangement modelled after UNIX-style shell programming. To avoid the need to deal with certain difficult types of SGML (e.g. minimised markup) texts are converted to a normal form before processing. A tool selects what information it requires from an input SGML stream and adds information as new SGML markup. LT XML is an extension of LT NSL to XML; in this case the normalisation step is unnecessary.

Other similar work in this area include: the XDOC workbench Rösner and Kunze (2002); Artola et al. (2002) report work on stand-off markup for NLP tools. Cassidy and Harrington (2001) discusses multi-level annotation of speech.

5.3.2 Reference Annotation (I): TIPSTER

The ARPA-sponsored TIPSTER programme in the US, which finished in 1998, produced a data-driven architecture for NLP systems Grishman (1997). A number of sites implemented the architecture, e.g., GATE version 1 Cunningham et al. (1999), ELLOGON Petasis et al. (2002); the initial prototype was written by Ted Dunning at the Computing Research Lab of New Mexico State University. In contrast to the embedding approach, in TIPSTER the text remains unchanged while information about it is stored in a separate database (DB). The database refers back to the text by means of offsets. The data is stored *by reference*.

Information is stored in the database in the form of *annotations*, which associate arbitrary information (*attributes*), with portions of documents (identified by sets of start/end character offsets or *spans*). Attributes will often be the result of linguistic analysis, e.g. POS tags. In this way information about texts is kept separate from the texts themselves. In place of an SGML DTD (or XML XSchema), an *annotation type declaration* defines the information present in annotation sets (though note that few implementations instantiated this part of the architecture). Figure 1 gives an example which "shows a single sentence and the result of three annotation procedures: tokenisation with part-of-speech assignment, name recognition, and sentence boundary recognition. Each token has a single attribute, its part of speech (pos), . . . ; each name also has a single attribute, indicating the type of name: person, company, etc." Grishman (1997).

Documents are grouped into *collections* (or corpora), each with an associated database storing annotations and document attributes such as identifiers, headlines, etc. The definition of documents and annotations in TIPSTER forms part of an Object-Oriented model that can deal with inter- as well as intra-textual information by means of reference objects that can point at annotations, documents and collections. The model also describes elements of IE and IR systems relating to their use, providing classes representing queries and information needs.

TIPSTER-style models have a number of advantages and disadvantages. Texts may appear to be one-dimensional, consisting of a sequence of characters, but this view is incompatible with structures like tables, which are inherently two-dimensional. Their representation and manipulation is easier in a referential model like TIPSTER than in an embedding one like SGML where markup is stored in a one-dimensional text string. In TIPSTER, a column of a table can be represented as a single object with multiple references to parts of the text (an *annotation* with multiple *spans*, or a document attribute with multiple references to annotations). Marking columns in SGML requires a tag for each row of the column, and manipulation of the structure as a whole necessitates traversal of all the tags and construction

Text

```
Kevin admired his bike.
0...|5...|10..|15..|20
```

Annotations

Id	Type	Span		Attributes
		Start	End	
1	token	0	5	pos=NP
2	token	6	13	pos=VBD
3	token	14	17	pos=PP
4	token	18	22	pos=NN
5	token	22	23	
6	name	0	5	name_type=person
7	sentence	0	23	

Figure 1. TIPSTER annotations example

of some other, non-SGML data structure.

Distributed control has a relatively straightforward implementation path in a database-centred system like TIPSTER: the DB can act as a blackboard, and implementations can take advantage of well-understood access control technology.

On the issue of storage overhead and contrasting with the hyperlinking used in LT XML, we see that:

(1) There is no need to break up a document into smaller chunks as the DBMS[10] in the document manager can deal efficiently with large data sets and visualisation tools can give intelligible views into this data.
(2) To cross-refer between annotations is a matter of citing ID numbers (which are themselves indexes into DB records and can be used for efficient data access). It is also possible to have implicit links: simple API calls will find all the token annotations subsumed by a sentence annotation, for example, via their respective byte ranges without any need for additional cross-referencing information.

Another disadvantage of embedded markup compared with TIPSTER is that an SGML structure like `<w id=p4.w1>` has to be parsed in order to extract the fact that we have a "w" tag whose "id" attribute is "p4.w1". A TIPSTER annotation is effectively a database record with separate fields for type (e.g. "w"), ID and other attributes, all of which may be indexed and none of which ever requires parsing.

[10] DataBase Management System.

There are three principal disadvantages of the TIPSTER approach.

(1) Editing of texts requires offset recalculation.
(2) TIPSTER specifies no interchange format, and TIPSTER data is weakly typed (there is no effective DTD mechanism; though this is also to an extent an advantage, as a complex typing scheme can inhibit unskilled users).
(3) The reference classes can introduce brittleness in face of changing data: unless an application chases all references and updates them in face of changes in the objects they point to, the data can become inconsistent.

(The latter problem also applies to hyperlinking in embedded markup.)

5.3.3 Reference Annotation (II): LDC

The Linguistic Data Consortium (LDC) have proposed the use of Directed Acyclic Graphs (DAGs), or just Annotation Graphs (AGs), as a unified data structure for text and speech annotation Bird et al. (2000b). Bird and Liberman (1999b) provides an example of using these graphs to to mark up discourse-level objects. This section compares the structure of TIPSTER annotations with the graph format.

TIPSTER annotations as discussed above are associated with documents and have:

- a type, which is a string;
- an ID, which is a string unique amongst annotations on the document;
- a set of spans that point into the text of the document;
- a set of attributes.

TIPSTER attributes, which are associated with annotations and with documents and collections of documents have:

- a name, which is a string;
- a value, which may be one of several data types including a string, a reference to annotation, document or collection, or a set of strings or references.

Some implementors of the architecture, including GATE and Corelli, have relaxed the type requirements on attribute values, allowing any object as a value. This has the advantage of flexibility and the disadvantage that viewing, editing and storage of annotations is made more complex.

TIPSTER explicitly models references between annotations with special reference classes. These rely on annotations, documents and collections of documents having unique identifiers.

LDC annotations are arcs in a graph whose nodes are time points (or, by extension, character offsets in a text). Each annotation has a type and a value, which are both atomic. A document may have a number of different graphs, and graphs can be associated with more than one document; this is not specified in the model.

There are no explicit references. References are handled implicitly by equivalence classes: if two annotations share the same type and value, they are considered coreferential. In order to refer to particular documents or other objects, an application or annotator must choose some convention for representing those references as strings, and use those as annotation values. This seems problematic: an annotation of type CoreferenceChain and value Chain23 should be equivalent to another of the same type and value, but this is not true for an annotation of type PartOfSpeech and value Noun.

LDC annotation values are atomic. Any representation of complex data structures must define its own reference structure to point into some other representation system.

TIPSTER is a richer formalism, both in terms of the complexity of the annotation/attribute part of the model and also because documents and collections of documents are an explicit part of the model, as are references between all these objects.

The inherent problems with developing a model of a task to be solved in software in isolation from the development of instances of that software are evident in Cassidy and Bird (2000), where the properties of the LDC AG model when stored and indexed in a relational database are discussed. At this point the authors added identifier fields to annotations to allow referencing without the equivalent class notion.

5.3.4 Reference Annotation (III): GATE

GATE version 2 has a reference annotation model that was designed to combine the advantages of the TIPSTER and LDC models:

- annotation sets are more explicitly graph-based. This allows increased efficiency of traversal and simpler editing because offsets are moved from the annotations into a separate node object. Also the offsets can be both character and time offsets, thus enabling annotation of multimodal data;
- multiple annotation sets are allowed on documents. Consider the situation when two people are adding annotation to the same document, and later wish to compare and merge their results. TIPSTER would handle this by having an "annotator" attribute on all the annotations. It is much simpler to have disjoint sets.
- documents and collections are an essential part of the model, and information can be associated with them in similar fashion to that on annotations;
- all annotations have unique identifiers to allow for referencing;
- an annotation only has two nodes, which means that the multiple-span annotations of TIPSTER are no longer supported (the workaround is to store non-contiguous data structures as features of the document, and point from there to the multiple annotations that make up the structures);
- the annotation values are extensible, i.e., any classes of object can be added to the model and be associated with annotations.

In addition, both LDC and TIPSTER need an annotation meta-language, to describe for purposes of validation or configuration of viewing and editing tools the structure and permissible value set of annotations. GATE uses the XML Schema language supported by W3C as an annotation meta-language Cunningham et al. (2002b). These annotation schemas define what attributes and optionally what values are permissible for each type of annotation (e.g., part-of-speech, named entity). For instance, a chosen tag set can be specified as permissible values for all part-of-speech annotations. This meta-information enables the annotation tools to control the correctness of the user input, thus making it easier to enforce annotation standards.

5.4 Data About Language

Preceding sections described language data – information related directly to examples of human performance of language. This section considers work on data about language, or description of human language competence. Much work in this area has concentrated on formalisms for the representation of the data, and has advocated declarative, constraint-based representations (using feature-structure matrices manipulated under unification) as an appropriate vehicle with which "many technical problems in language description and computer manipulation of language can be solved" Shieber (1992). One example of an infrastructure project based on Attribute-Value Matrices (AVMs) is ALEP – the Advanced Language Engineering Platform. ALEP aims to provide "the NLP research and engineering community in Europe with an open, versatile, and general-purpose development environment" Simkins (1992, 1994); Eriksson (1996). ALEP, while in principle open, is primarily an advanced system for developing and manipulating feature structure knowledge-bases under unification. Also provided are several parsing algorithms, algorithms for transfer, synthesis and generation Schütz (1994). As such, it is a system for developing particular types of LR (e.g. grammars, lexicons) and for *doing* a particular set of tasks in LE in a particular way.

The system, despite claiming to use a theory-neutral formalism (in fact an HPSG-like formalism), is still committed to a particular approach to linguistic analysis and representation. It is clearly of utility to those in the LE community using that class of theories and to whom those formalisms are relevant; but it excludes, or at least does not actively support, those who are not, including an increasing number of researchers committed to statistical and corpus-based approaches.

Other systems that use AVMs include: Zajac (1992), a framework for defining NLP systems based on AVMs; the Eurotra architecture Schütz et al. (1991): an 'open and modular' architecture for MT promoting resource reuse; the DATR morphological lexicon formalism Evans and Gazdar (1996); the Shiraz MT Architecture Amtrup (1999), a chart and unification-based architecture for MT; Zajac (1998a), a unified FST/AVM formalism for morphological lexicons; the RAGS architecture (see below).

A related issue is that of grammar development in an LE context – see Netter and Pianesi (1997); Estival et al. (1997). Fischer et al. (1996) presents an abstract model of Thesauri and terminology maintenance in an OO framework. ARIES Goni et al. (1997) is a formalism and development tool for Spanish morphological lexicons.

The RAGS (Reference Architecture for Generation Systems) project Cahill et al. (1999b,a) has concentrated on describing structures that may be shared amongst NLG component interfaces. This choice is motivated by the fact that the input to a generator is not a document but a meaning representation. RAGS describes component I/O using a nested feature matrix representation, but doesn't describe the types of LR that an NLG system may use, or the way in which components may be represented, loaded, etc. More recently Mellish et al. (2004) present the RAGS conceptual framework, while Mellish and Evans (2004) discuss the implementation of this framework in several experimental systems, and how these systems illustrate a range of wider issues for the construction of SALE for generation.

5.5 Indexing and Retrieval

Modern corpora, and annotations upon them, frequently run to many millions of tokens. To enable efficient access to this data the tokens and annotation structures must be indexed. In the case of raw corpora, this problem equates to Information Retrieval (IR; aka document detection), a field with a relatively well-understood set of techniques based on treating documents as bags of stemmed words and retrieving based on relative frequency of these terms in documents and corpora – see for example van Rijsbergen (1979). Although these processes are well understood and relatively static, nevertheless IR is an active research field, partly because existing methods are imperfect and partly because that imperfection becomes more and more troubling in face of the explosion of the Web. There have been a number of attempts to provide SALE systems in this context.

As noted above, the TIPSTER programme developed a reference model of typical IR component sets TIPSTER (1995). More concretely, this programme also developed a communication protocol based on Z39.50 for detection interactions between querying application and search engine Buckley (1998). The annotation and attribute data structures described in section 5.3.2 were also applied for IR purposes, though the practical applications of the architecture were found in general to be too slow for the large data sets involved.

GATE Cunningham et al. (2002b,a) uses an extendable, open-source IR engine, Lucene, in order to index documents and corpora for full-text retrieval. Lucene also allows indexing and retrieval by custom-provided fields like annotations. The model used to wrap Lucene in GATE is designed for extensibility to other IR systems when required.

Whereas the problem of indexing and retrieving documents is well understood, the problem of indexing complex structures in annotations is more of an open question. The Corpus Query System Christ (1994, 1995) is the most cited source in this area, providing indexing and search of corpora and later of WordNet. Similar ideas have been implemented in CUE Mason (1998) for indexing and search of annotated corpora, and at the W3-Corpora site University of Essex (1999) for searchable on-line annotated corpora. Some work on indexing in the LT XML system is reported in McKelvie and Mikheev (1998). Bird et al. (2000a) propose a query language for the LDC annotation graph model, called AGQL. Cassidy (2002) discusses the use of XQuery as an annotation query language and concludes that it is good for dealing with hierarchical data models like XML, but needs extending with better support for sequential data models like annotation graphs.

GATE indexes and retrieves annotations by storing them in a relational database, indexed by type, attributes and their values. In this way, it is possible to retrieve all documents that contain a given attribute and/or value or retrieve all annotations of a given type in a corpus, without having to traverse each document separately Cunningham et al. (2002b); Bontcheva et al. (2002). The query language used is SQL.

6 Recent Trends and Future Directions

As has become evident from the work reviewed here, there are many tools and architectures, many of them focused on sub-areas of NLP (e.g., dialog, speech) or specific formalisms (e.g. HPSG). Each of these infrastructures offers specialised solutions, so it is not likely that there will ever be only one, universal architecture or infrastructure. Instead, the focus in recent work has been on *inter-operability* – allowing infrastructures to work together – and *reusability* – enabling users to reuse and adapt tools with a minimum effort. We will review some of these new trends here to see how they are likely to influence the next period of research on SALE.

6.1 Towards multipurpose repositories

In order to support reusability of resources, a number of repositories have been established, some describing NLP tools e.g., ACL Natural Language Software Registry, and others distributing Language Resources like corpora and lexicons, e.g, ELRA and LDC. To date these repositories have remained largely independent of each other, with the exception of repositories like TRACTOR Martin (2001), which contain both corpora in a number of languages and specialised tools for corpus analysis.

As argued in Declerck (2001), there is a need for linking the two kinds of repositories in order to allow corpus researchers find the tools they need to process corpora and vice versa. The idea is to create a multipurpose *infrastructure* for the storage and access of both language data and the corresponding processing resources. One of the cornerstones of such an infrastructure is metadata, associated with each resource and pointing at other relevant resources, e.g., tools pointing at the language data that they need and can process. In the following section, we discuss recent research on metadata descriptions for tools and Language Resources, including handling of multimodal and multilingual data.

6.2 Resource Metadata and Annotation Standards

As discussed in section 4.3, there are a number of reasons why metadata should be part of a component infrastructure, i.e., why it is useful beyond the more narrow scope of providing descriptions of resources in a repository. One dimension that affects the kinds of metadata needed to describe resources is their type, e.g., whether they are documents in a corpus, a lexicon, or a tool working on language data. For example, the ISLE Computational Lexicon working group has defined a modular architecture, called MILE, designed to factor out linguistically independent primitive units of lexical information, deal with monolingual, bilingual and multilingual lexicons, and avoid theoretical bias Calzolari et al. (2001). Some of these desiderata are relevant also to the problem of resource distribution, as discussed in section 5.1 and Cunningham et al. (2000). Multimedia/multimodal Language Resources (MMLR) pose a different set of problems and existing standards for tagging textual documents (e.g., XCES Ide et al. (2000)) are not sufficient. Broeder and Wittenburg (2001) provide a metadata vocabulary for MMLR, which encodes information related to the media files (e.g., format and size) and the annotation units used (e.g., part-of-speech), as well as the basic information on creator, content, etc.

Another aspect of improving resource reusability and interoperability is the development of standards for encoding annotation data. Ide and Romary (2002) describes a framework for linguistic annotations based on XML and the XML-based RDF and DAML+OIL standards for defining the semantics of the annotations. This provides a link with recent work on formal ontologies and the Semantic Web and enables the use of the related knowledge management tools to support linguistic annotation. For example, Collier et al. (2002) use the popular Protégé ontology editor as a basis of an annotation tool capable of producing RDF(S) annotations of language data in multiple languages.

6.3 Open Archives

One of the new research directions has been towards *open archives* – archives aiming to make resources easily discoverable, accessible and identifiable. This not only includes Language Resources like corpora and lexicons but also software tools, i.e. processing resources and development environments. Resource discovery is made possible by meta-data associated with each resource and made available in a centralised repository. The recently established Open Language Archives Community (OLAC) Bird et al. (2002); Bird and Simons (2001) aims at creating a world-wide virtual library of Language Resources, through development of interoperating repositories and tools for their maintenance and access. OLAC also aims at establishing and promoting best practice in archiving for Language Resources. The OLAC infrastructure is based on two initiatives from digital library research – the Open Archives Initiative (`www.openarchives.org`) and the Dublin Core initiative for resource metadata (`dublincore.org`). Currently OLAC comprises 12 archives with a cross-archive searching facility.

As argued in Wynne (2002), the current trends towards multilinguality and multimodality suggest that the Language Resources of the future will span across languages and modalities, will be distributed over many repositories and form virtual corpora, supported by a diverse set of linguistic analysis and searching tools. As already discussed in the previous section, metadata and annotation standards play a very important role here. The other major challenge lies in making existing processing resources accessible over the Web and enhance their reusability and portability. This is where we look next.

6.4 Component Reusability, Distributed Access and Execution

In order to enable virtual corpora and collaborative annotation efforts spanning country boundaries, software infrastructures and tools need to control user access to different documents, types of annotations, and metadata. Ma et al. (2002) discuss how this can be achieved by using a shared relational database as a storage medium, combined with a number of annotation tools based on the annotation graph formalism discussed in section 5.3.3. The same approach has been taken in GATE Cunningham et al. (2002b) where all Language Resources and their associated annotations can be stored in Oracle or PostgreSQL. This enables users to access remote LRs, index LRs by their annotations, and construct search queries retrieving LRs given annotations or metadata constraints (e.g., find all documents that contain person entities called Bush). User access is controlled at individual and group level, with read/write access rights specified at LR creation time by their owner (the user who has first stored the LR in the database). Since the storage mechanisms in GATE are separate from the API used for accessing LRs and annotations, the visualisation tools and processing resources work on both local and remote data in the same way. Ma et al. (2002) discuss a special version of AGTK TableTrans tool created to work with the database annotations. In addition, GATE's database storage model supports other LRs like lexicons and ontologies.

The recent development of web services enables integration of different information repositories and services across the Internet and offers a new way of sharing Language Resources across the Internet. Dalli (2002) discusses an architecture for Web-based interoperable LRs based on SOAP and Web services. (Work in progress extends this approach to processing resource execution in the context of on-line adaptive Information Extraction – see Tablan et al. (2003).) Both make extensive use of XML for metadata description. However, the benefits of the relational database storage mechanism can still be maintained by providing a conversion layer, which transforms the stored LRs and annotations into the desired XML format when needed. Similarly, Todirascu et al. (2002) describe an architecture that uses SOAP to provide distributed processing resources as services on the net, both as a protocol for message passing and a mechanism for executing remote modules from the client.

Bontcheva et al. (2004) report recent work in upgrading GATE to meet challenges posed by research in semantic web, large-scale digital libraries and machine learning for language analysis.

Popov et al. (2004) present an application that combines several SALE systems, including

GATE and Sesame[11], to create a platform for semantic annotation called KIM (Knowledge and Information Management). The paper covers a number of issues relating to architecting scaleable ontology-based Information Extraction.

6.5 Measurement

A persistent theme of SALE work has been measurement, quantitative evaluation and the relationship between engineering practice and scientific theory. To quote Kelvin:

> When you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind: it may be the beginning of knowledge, but you have scarcely in your thoughts advanced to the stage of science. (Lord Kelvin (William Thomson), in a lecture to the Institution of Civil Engineers, London, 3 May 1883.)

On the other hand, Einstein tells us:

> Not everything that counts can be counted, and not everything that can be counted counts. (Albert Einstein, from a sign hanging in his office at Princeton University.)

Researchers have taken similarly varied approaches to measurement, both of component systems developed using SALE systems and of the success of those systems themselves. The presentation of IBM's TEXTRACT architecture by Neff et al. (2004) includes an illustration of how the same mechanism can be used for producing both quantitative metrics and for visual feedback to users of the results of automated processing.

Ferrucci and Lally (2004) report a successor to TEXTRACT called UIMA (Unstructured Information Managment Architecture), which is in active development to support the work of several hundred research and development staff working in areas as diverse as question answering and machine translation. The significant commitment of IBM to SALE development indicates the success of the TEXTRACT concept, and of architectural support for language processing research.

7 Prognosis

The principal defining characteristic of NLE work is its objective: to engineer products which deal with natural language and which satisfy the constraints in which they have to operate. This definition may seem tautologous or a statement of the obvious to an engineer practising in another, well established area (e.g. mechanical or civil engineering), but is still a useful reminder to practitioners of software engineering, and it becomes

[11] `http://www.openrdf.org/`

near-revolutionary when applied to natural language processing. This is partly because of what, in our opinion, has been the ethos of most Computational Linguistics research. Such research has concentrated on studying natural languages, just as traditional Linguistics does, but using computers as a tool to model (and, sometimes, verify or falsify) fragments of linguistic theories deemed of particular interest. This is of course a perfectly respectable and useful scientific endeavour, but does not necessarily (or even often) lead to working systems for the general public. Boguraev et al. (1995.)

Working systems for public consumption require qualities of robustness that are unlikely to be achieved at zero cost as part of the normal development of experimental systems in language computation research Maynard et al. (2002). Investing the time and energy necessary to create robust reusable software is not always the right thing to do, of course – sometimes what is needed is a quick hack to explore some simple idea with as little overhead as possible. To conclude that this is always the case is a rather frequent error, however, and this is of particular concern at a time when web-scale challenges to language processing are common.

Also problematic for SALE is the fact that it is not always easy to justify the costs of engineered systems when developers of more informal and short-term solutions have been known to make claims for their power and generality that are, shall we say, somewhat optimistic. [12] . The fact that the majority of the language processing field has, as we write this in the mid-naughties, used a SALE system of one type or another indicates that this has been a fruitful pursuit.

References

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

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

Artola, X., de Ilarraza, A. D., Ezeiza, N., Gojenola, K., Hernández, G., Soroa, A., 2002. A Class Library for the Integration of NLP Tools: Definition and implementation of an Abstract Data Type Collection for the manipulation of SGML documents in a context of stand-off linguistic annotation. In: Proceedings of LREC 2002 Third International Conference on Language Resources and Evaluation. Las Palmas, Canary Islands - Spain, pp. 1650–1657.

Berners-Lee, T., Connolly, D., Swick, R., 1999. Web Architecture: Describing and Exchanging Data. Tech. rep., W3C Consortium, `http://www.w3.org/1999/04/WebData`.

Bird, S., Buneman, P., Tan, W., 2000a. Towards a query language for annotation graphs. In:

[12] How to revolutionise distributed programming, the easy way: **Monday**: 'I heard someone mention the Grid. What is this Grid thing?' **Tuesday** (to PhD student): 'Can you put a SOAP interface on that?' **Wednesday** (giving a talk): 'Our IE system is fully distributed, with a Grid-based architecture.'

Proceedings of the Second International Conference on Language Resources and Evaluation. Athens.

Bird, S., Day, D., Garofolo, J., Henderson, J., Laprun, C., Liberman, M., 2000b. ATLAS: A flexible and extensible architecture for linguistic annotation. In: Proceedings of the Second International Conference on Language Resources and Evaluation. Athens.

Bird, S., Liberman, M., 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., Liberman, M., 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. pp. 1–10.

Bird, S., Simons, G., 2001. The olac metadata set and controlled vocabularies. In: Proceedings of the ACL 2001 Workshop on Sharing Tools and Resources. pp. 27–38.

Bird, S., Uszkoreit, H., Simons, G., 2002. The open language archives community. In: Proceedings of 3rd Language Resources and Evaluation Conference (LREC'2002). Gran Canaria, Spain.

Boguraev, B., Garigliano, R., Tait, J., 1995. Editorial. Natural Language Engineering. 1, Part 1.

Boitet, C., Seligman, M., 1994. The "Whiteboard" Architecture: A Way to Integrate Heterogeneous Components of NLP Systems. In: Proceedings of COLING '94. Kyoto, Japan, pp. 426–430.

Bontcheva, K., Cunningham, H., Tablan, V., Maynard, D., Saggion, H., 2002. Developing Reusable and Robust Language Processing Components for Information Systems using GATE. In: Proceedings of the 3rd International Workshop on Natural Language and Information Systems (NLIS'2002). IEEE Computer Society Press, Aix-en-Provence, France.

Bontcheva, K., Tablan, V., Maynard, D., Cunningham, H., 2004. Evolving GATE to Meet New Challenges in Language Engineering. Natural Language Engineering 10 (3/4), 349—373.

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

Bos, J., Rupp, C., Buschbeck-Wolf, B., Dorna, M., 1998. Managing information at linguistic interfaces. In: Proceedings of the 36th ACL and the 17th COLING (ACL-COLING '98). Montreal, pp. 160–166.

Brand, S., 1994. How Buildings Learn. Penguin, London.

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

Brill, E., 1992. A simple rule-based part-of-speech tagger. In: Proceedings of the Third Conference on Applied Natural Language Processing. Trento, Italy.

Broeder, D., Wittenburg, P., 2001. Multimedia language resources. In: Proceedings of the ACL 2001 Workshop on Sharing Tools and Resources. pp. 47–51.

Brown, P., Cocke, J., Pietra, S. D., Pietra, V. D., Jelinek, F., Lafferty, J., Mercer, R., Roossin, P., 1990. A Statistical Approach to Machine Translation. Computational Linguistics 16 (June), 79–85.

Brugman, H., Russel, A., Wittenburg, P., Piepenbrock, R., 1998a. Corpus-based Research using the Internet. In: Workshop on Distributing and Accessing Linguistic Resources. Granada, Spain, pp. 8–15, `http://www.dcs.shef.ac.uk/~hamish/dalr/`.

Brugman, H., Russel, H., Wittenburg, P., 1998b. 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.

Burnard, L., May 1995. Users Reference Guide for the British National Corpus, `http://info.ox.ac.uk/bnc/`.

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

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

Cahill, L., Doran, C., Evans, R., Paiva, D., Scott, D., Mellish, C., Reape, M., Apr. 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. The Society for the Study of Artificial Intelligence and Simulation of Behaviour, Edinburgh.

Cahoon, B., McKinley, K., 1996. Performance Evaluation of a Distributed Architecture for Information Retrieval. In: Proceedings of SIGIR '96. Zurich, pp. 110–118.

Calzolari, N., Lenci, A., Zampolli, A., 2001. International standards for multilingual resource sharing: The isle computational lexicon working group. In: Proceedings of the ACL 2001 Workshop on Sharing Tools and Resources. pp. 39–46.

Carreras, X., Padró, L., 2002. A Flexible Distributed Architecture for Natural Language Analyzers. In: Proceedings of the LREC 2002 Third International Conference on Language Resources and Evaluation. Las Palmas, Canary Islands - Spain, pp. 1813–1817.

Cassidy, S., 2002. Xquery as an annotation query language: a use case analysis. In: Proceedings of 3rd Language Resources and Evaluation Conference (LREC'2002). Gran Canaria, Spain.

Cassidy, S., Bird, S., 2000. Querying databases of annotated speech. In: Eleventh Australasian Database Conference. Australian National University, Canberra.

Cassidy, S., Harrington, J., 2001. Multi-level annotation in the Emu speech database management system. Speech Communication 33, 61–77.

Cheong, T., Kwang, A., Gunawan, A., Loo, G., Qwun, L., Leng, S., 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, pp. B371–B377.

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). Budapest, `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.

Clements, P., Northrop, L., 1996. Software Architecture: An Executive Overview. Tech. Rep. CMU/SEI-96-TR-003, Software Engineering Institute, Carnegie Mellon University.

Collier, N., Takeuchi, K., Nobata, C., Fukumoto, J., Ogata, N., 2002. Progress on multilingual named entity annotation guidelines using rdf(s). In: Proceedings of the 3rd Language Resources and Evaluation Conference (LREC'2002). Gran Canaria, Spain.

Cunningham, H., 1999. A Definition and Short History of Language Engineering. Journal of Natural Language Engineering 5 (1), 1–16.

Cunningham, H., 2000. Software Architecture for Language Engineering. Ph.D. thesis, University of Sheffield, `http://gate.ac.uk/sale/thesis/`.

Cunningham, H., 2002. GATE, a General Architecture for Text Engineering. Computers and the Humanities 36, 223–254.

Cunningham, H., Bontcheva, K., Peters, W., Wilks, Y., September 2000. Uniform language resource access and distribution in the context of a General Architecture for Text Engineering (GATE). In: Proceedings of the Workshop on Ontologies and Language Resources (OntoLex'2000). Sozopol, Bulgaria, `http://gate.ac.uk/sale/ontolex/ontolex.ps`.

Cunningham, H., Freeman, M., Black, W., 1994. Software Reuse, Object-Oriented Frameworks and Natural Language Processing. In: New Methods in Language Processing (NeMLaP-1), September 1994. (Re-published in book form 1997 by UCL Press), Manchester.

Cunningham, H., Gaizauskas, R., Humphreys, K., Wilks, Y., Apr. 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. The Society for the Study of Artificial Intelligence and Simulation of Behaviour, Edinburgh.

Cunningham, H., Humphreys, K., Gaizauskas, R., Wilks, Y., Mar. 1997. Software Infrastructure for Natural Language Processing. In: Proceedings of the 5th Conference on Applied Natural Language Processing (ANLP-97). `http://xxx.lanl.gov/abs/cs.CL/9702005`.

Cunningham, H., Maynard, D., Bontcheva, K., Tablan, V., 2002a. GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications. In: Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics (ACL'02).

Cunningham, H., Maynard, D., Bontcheva, K., Tablan, V., Ursu, C., 2002b. The GATE User Guide. `http://gate.ac.uk/`.

Cunningham, H., Peters, W., McCauley, C., Bontcheva, K., Wilks, Y., 1998. 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., Scott, D., 2004. Introduction to the Special Issue on Software Architecture for Language Engineering. Natural Language Engineering.

Dalli, A., 2002. Creation and evaluation of extensible language resources for maltese. In: Proceedings of 3rd Language Resources and Evaluation Conference (LREC'2002). Gran Canaria, Spain.

Declerck, T., 2001. Introduction: Extending nlp tools repositories for the interaction with language data resource repositories. In: Proceedings of the ACL 2001 Workshop on Sharing Tools and Resources. pp. 3–6.

DFKI, 1999. The Natural Language Software Registry, `http://www.dfki.de/lt/registry/`.

EAGLES, 1999. EAGLES recommendations, `http://www.ilc.pi.cnr.it/-EAGLES96/browse.html`.

Edmondson, W., Iles, J., 1994. A Non-linear Architecture for Speech and Natural Language Processing. In: Proceedings of International Conference on Spoken Language Processing (ICSLP '94). Vol. 1. Yokohama, Japan, pp. 29–32.

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

Erman, L., Hayes-Roth, F., Lesser, V., Reddy, D., 1980. The Hearsay II speech understanding system: integrating knowledge to resolve uncertainty. Computing Surveys 12.

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

Evans, R., Gazdar, G., 1996. DATR: A Language for Lexical Knowledge Representation. Computational Linguistics 22 (1).

Ferrucci, D., Lally, A., 2004. UIMA: An Architectural Approach to Unstructured Information Processing in the Corporate Research Environment. Natural Language Engineering.

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

Fischer, D., Mohr, W., Rostek, L., 1996. A Modular, Object-Oriented and Generic Approach for Building Terminology Maintenance Systems. In: TKE '96: Terminology and Knowledge Engineering. Frankfurt, pp. 245–258.

Gaizauskas, R., Wilks, Y., 1998. Information Extraction: Beyond Document Retrieval. Journal of Documentation 54 (1), 70–105.

Goldfarb, C., Prescod, P., 1998. The XML Handbook. Prentice Hall, New York.

Goldfarb, C. F., 1990. The SGML Handbook. Oxford University Press.

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

Görz, G., Kessler, M., Spilker, J., Weber, H., 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. Tech. rep., DARPA, `http://www.itl.nist.gov/div894/894.02/related_projects/tipster/`.

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

Herzog, G., Ndiaye, A., Merten, S., Kirchmann, H., Becker, T., Poller, P., 2004. Large-Scale Software Integration for Spoken Language and Multimodal Dialog Systems. Natural Language Engineering.

Hobbs, J., 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., Cummins, F., 1989. TARO: An Interactive, Object-Oriented Tool for Building Natural Language Systems. In: IEEE International Workshop on Tools for Artificial Intelligence. Los Angeles, pp. 108–113.

Ide, N., 1998. Corpus Encoding Standard: SGML Guidelines for Encoding Linguistic Corpora. In: Proceedings of the First International Language Resources and Evaluation Conference. Granada, Spain, pp. 463–470.

Ide, N., Bonhomme, P., Romary, L., 2000. XCES: An XML-based Standard for Linguistic Corpora. In: Proceedings of the Second International Language Resources and Evaluation Conference (LREC). Athens, Greece, pp. 825–830.

Ide, N., Romary, L., 2002. Standards for language resources. In: Proceedings of 3rd Language Resources and Evaluation Conference (LREC'2002). Gran Canaria, Spain.

Ide, N., Romary, L., 2004. Standards for language resources. Natural Language Engineering.

Jing, H., McKeown, K., 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, pp. 607–613.

Kay, M., Gawron, J., Norvig, P., 1994. Verbmobil, A Translation System for Face-to-Face Dialog. CSLI, Stanford, CA.

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

Lassila, O., Swick, R., 1999. Resource Description Framework (RDF) Model and Syntax Specification. Tech. Rep. 19990222, W3C Consortium, `http://www.w3.org/-TR/REC-rdf-syntax/`.

Lavelli, A., Pianesi, F., Maci, E., Prodanof, I., Dini, L., Mazzini, G., 2002. SiSSA: An infrastructure for developing nlp applications. In: Proceedings of 3rd Language Resources and Evaluation Conference (LREC'2002). Gran Canaria, Spain.

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

LuperFoy, S., Loehr, D., Duff, D., Miller, K., Reeder, F., Harper, L., 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, pp. 794–801.

Ma, X., Lee, H., Bird, S., Maeda, K., 2002. Models and tools for collaborative annotation. In: Proceedings of 3rd Language Resources and Evaluation Conference (LREC'2002). Gran Canaria, Spain.

Macleod, C., Ide, N., Grishman, R., 2002. The American National Corpus: Standardized Resources for American English. In: Proceedings of 2nd Language Resources and Evaluation Conference (LREC). Athens, Greece, pp. 831–836.

Marcus, M., Santorini, B., Marcinkiewicz, M., 1993. Building a Large Annotated Corpus of English: The Penn Treebank. Computational Linguistics 19 (2), 313–330.

Martin, W., 2001. An archive for all of europe. In: Proceedings of the ACL 2001 Workshop on Sharing Tools and Resources. pp. 11–14.

Mason, O., 1998. The CUE Corpus Access Tool. In: Workshop on Distributing and Accessing Linguistic Resources. Granada, Spain, pp. 20–27, `http://www.dcs.shef.ac.uk/-~hamish/dalr/`.

Maynard, D., Tablan, V., Cunningham, H., Ursu, C., Saggion, H., Bontcheva, K., Wilks, Y., 2002. Architectural Elements of Language Engineering Robustness. Journal of Natural Language Engineering – Special Issue on Robust Methods in Analysis of Natural Language Data 8 (2/3), 257–274.

McClelland, J., Rumelhart, D., 1986. Parallel Distributed Processing. MIT Press, Cambridge, MA.

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

McKelvie, D., Mikheev, A., 1998. Indexing SGML files using LT NSL, lT Index documentation, from `http://www.ltg.ed.ac.uk/`.

Mellish, C., Evans, R., 2004. Implementation Architectures for Natural Language Generation. Natural Language Engineering.

Mellish, C., Scott, D., Apr. 1999. Workshop preface. In: Proceedings of the AISB'99 Workshop on Reference Architectures and Data Standards for NLP. The Society for the Study of Artificial Intelligence and Simulation of Behaviour.

Mellish, C., Scott, D., Cahill, L., Evans, R., Paiva, D., Reape, M., 2004. A Reference Architecture for Generation Systems. Natural Language Engineering.

Miller (Ed.), G. A., 1990. WordNet: An on-line lexical database. International Journal of Lexicography 3 (4), 235–312.

MITRE, 2002. Galaxy Communicator, `http://communicator.sourceforge.net/`.

Neff, M. S., Byrd, R. J., Boguraev, B. K., 2004. The Talent System: TEXTRACT Architecture and Data Model. Natural Language Engineering.

Nelson, T., 1997. Embedded Markup Considered Harmful. In: Connolly, D. (Ed.), XML: Principles, Tools and Techniques. O'Reilly, Cambridge, MA, pp. 129–134.

Netter, K., Pianesi, F., 1997. Preface. In: Proceedings of the Workshop on Computational Environments for Grammar Development and Linguistic Engineering. Madrid, pp. iii–v.

Ogden, B., 1999. TIPSTER annotation and the Corpus Encoding Standard, `http://crl.nmsu.edu/Research/Projects/tipster/annotation`.

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

Petasis, G., Karkaletsis, V., Paliouras, G., Androutsopoulos, I., Spyropoulos, C., 2002. Ellogon: A new text engineering platform. In: Proceedings of 3rd Language Resources and Evaluation Conference (LREC'2002). Gran Canaria, Spain.

Peters, W., Cunningham, H., McCauley, C., Bontcheva, K., Wilks, Y., 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`.

Popov, B., Kiryakov, A., Kirilov, A., Manov, D., Ognyanoff, D., Goranov, M., 2004. KIM – Semantic Annotation Platform. Natural Language Engineering.

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

Reiter, E., Dale, R., 2000. Building Natural Language Generation Systems. Cambridge University Press, Cambridge.

Rösner, D., Kunze, M., 2002. An XML-based document suite. In: Proceedings of the 19th International Conference on Computational Linguistics (COLING'02). Taipei, Taiwan.

Schütz, J., Oct. 1994. Developing Lingware in ALEP. ALEP User Group News, CEC Luxemburg 1 (1).

Schütz, J., Thurmair, G., Cencioni, R., 1991. An Architecture Sketch of Eurotra-II. In: MT Summit III. Washington D.C., pp. 3–11.

Shieber, S., 1992. Constraint-Based Grammar Formalisms. MIT Press, Cambridge, MA.

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.

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

Sperberg-McQueen, C., Burnard, L. (Eds.), 2002. Guidelines for Electronic Text Encoding and Interchange (TEI P4). The TEI Consortium.

Tablan, V., Bontcheva, K., Maynard, D., Cunningham, H., 2003. OLLIE: On-Line Learning for Information Extraction. In: Proceedings of the HLT-NAACL Workshop on Software Engineering and Architecture of Language Technology Systems. Edmonton, Canada.

Tablan, V., Ursu, C., Bontcheva, K., Cunningham, H., Maynard, D., Hamza, O., McEnery, T., Baker, P., Leisher, M., 2002. A Unicode-based Environment for Creation and Use of Language Resources. In: 3rd Language Resources and Evaluation Conference.

The Object Management Group, 1992. The Common Object Request Broker: Architecture and Specification. John Wiley, New York.

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

Todirascu, A., Kow, E., Romary, L., 2002. Towards reusable nlp components. In: Proceedings of 3rd Language Resources and Evaluation Conference (LREC'2002). Gran Canaria, Spain.

Tracz, W., Mar. 1995. Domain-Specific Software Architecture (DSSA) Frequently Asked Questions (FAQ), `http://www.oswego.com/dssa/faq/faq.html`.

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

van Rijsbergen, C., 1979. Information Retrieval. Butterworths, London.

Veronis, J., Ide, N., Apr. 1996. Considerations for the Reusability of Linguistic Software. Tech. rep., EAGLES, `http://w3.lpl.univ-aix.fr/projects/multext/LSD/LSD1.html`.

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

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

Wynne, M., 2002. The language resource archive of the 21st century. In: Proceedings of 3rd Language Resources and Evaluation Conference (LREC'2002). Gran Canaria, Spain.

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

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

Zajac, R., 1992. Towards Computer-Aided Linguistic Engineering. In: Proceedings of COLING '92. Nantes, France, pp. 828–834.

Zajac, R., 1997. An Open Distributed Architecture for Reuse and Integration of Heterogenous NLP Components. In: Proceedings of the 5th conference on Applied Natural Language Processing (ANLP-97).

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, Spain, pp. 34–40, `http://www.dcs.shef.ac.uk/~hamish/dalr/`.