Composing a General-Purpose Toolbox for Swedish

Fredrik Olsson and **Björn Gambäck**
{fredriko,gamback}@sics.se

Information and Language Engineering Group
Swedish Institute of Computer Science
Box 1263, S-164 29 Kista, Sweden
http://www.sics.se/humle/ile

Abstract

The paper discusses the lessons we have learned from the work on building a reusable toolset for Swedish within the framework of GATE, the General Architecture for Text Engineering, from the University of Sheffield, UK.

We describe our toolbox SVENSK and the reasons behind the choices made in the design, as well as the overall conclusions for language processing toolbox design which can be drawn.

1 Introduction

Why is it desirable to have a general-purpose toolset for Language Engineering? In general, it is likely that the following items hold:

language diversity
: Research in Language Engineering tends to be expensive since the results may not always be shared across languages, e.g., a tagger or parser for German is not applicable to Swedish. This implies that much of the work carried out in one language has to be carried out in other languages as well.

evaluation
: Evaluation of language processing software is a cumbersome task, and it would ease up things if researchers could cooperate in constructing test-suits and measures that apply to those and then share data and methods within a common framework.

commercialisation
: If you want to go commercial, it is important that the prototyping and testing phases can be carried out without the overhead of having to construct a new framework each time a new kind of system is to be developed.

In addition, for small languages like Swedish (with about 9 million speakers), there are not that many researchers in Computational Linguistics, and thus not many at all in the various sub-fields of the area. To be able to share results between groups in the same research area is crucial for every-day research. Both to show off results and for teaching purposes.

In order to encompass this, a general framework for Language Engineering could, or *should*, be expected to:

- cut development time and cost by reusing what has been done before;
- ensure that systems are scalable to prevent unexpected draw-backs due to the "toy problem syndrome";
- provide, in the long run, a good setting for evaluation of language engineering tasks.

In this paper we will discuss the lessons we have learned from the work on building such a toolbox for Swedish; however, we set out by describing some of the reasons for why our project from the start was laid out as it was.

In particular, the section following concentrates on our own project and toolbox architecture together with University of Sheffield's underlying GATE system, while Section 3 draws on the experiences from some other, previous and current, work on designing large language processing systems. Section 4 then moves on to the central purpose of the paper, discussing the main insights gained during the course of the project. Finally, Section 5 supplies a short bullet list with some of the overall conclusions we have drawn.

2 A toolbox for Swedish: SVENSK

The SVENSK project (Eriksson and Gambäck, 1997; Olsson et al., 1998; Gambäck and Olsson, 2000) is a national effort funded by the Swedish National Board for Industrial and Technical Development (NUTEK) to encompass some of the difficulties outlined above. The aim of SVENSK has been to develop a multi-purpose language processing system for Swedish based, where possible, on existing components, and targeted at research and teaching. The SVENSK system as such is thus mainly the sum of a fairly large set of different reusable language resources.

2.1 Choice of platform

In 1995 when the SVENSK project started to take shape, there was a need for a platform flexible enough to act as a framework for the language processing programs intended to constitute the toolbox. At the end of the selection process, there were two platforms remaining; the European Commission initiative ALEP, Advanced Language Engineering Platform (Simpkins and Groenendijk, 1994) and GATE, General Architecture for Text Engineering, from the University of Sheffield, UK (Cunningham et al., 1996).

GATE was chosen since it was, among other things, freely available and did not impose its own linguistic theories on the modules to be integrated. Even though ALEP, at the time, turned out to be too slow to fit as a software framework for SVENSK, it was considered feasible to integrate external modules in it (Eriksson and Gambäck, 1997).

More general points about both these projects will be discussed below, ALEP in Section 3.3 and GATE in Section 3.5. We will right away describe the GATE system as such, though, from our perspective within the SVENSK project.

2.2 GATE

GATE consists of three different parts; a document manager, a graphical interface, and a set of language engineering objects. This section gives an overview of each one of them.

2.2.1 GATE Document Manager, GDM

The GDM, which is based on the TIPSTER database architecture (Grishman and others, 1997), serves as a communication center for the components in GATE. It stores all information about texts that language processing systems require to run, as well as the information they produce. The GDM stores annotations associated to sequences of byte offsets in the original text. Each annotation may have several attributes, which in turn may have zero or more values. The byte offsets are used as pointers into the original text in order to enable separate storage of the source text and the database holding information associated to it. Also, the GDM provides a well-defined application programming interface (API) for handling the data it stores.

2.2.2 GATE Graphical Interface, GGI

The GGI is a graphical launch-pad which enables interactive testing and building of language processing systems within GATE. Various tasks are supported, such as integrating new language processing modules, building systems, launching them, and viewing the results. The philosophy of the interface is to provide the user woth a rich set of tools. There are, for example, several generic viewers for displaying module results, ranging from raw annotations to complex parse trees via output from part-of-speech taggers.

2.2.3 Language engineering objects

At the very heart of building a GATE-based system system we find the so-called Collection of REusable Objects for Language Engineering, CREOLE. The CREOLE modules/objects in the GATE system should be thought of as interfaces to resources; data, algorithmical or a mixture of both. A CREOLE module may be a "wrapper" around an already existing piece of software or it may be an entire program developed explicitly for GATE compliance. It is the CREOLE modules that perform the real work of analysing texts in a GATE-based system.

The tasks for a CREOLE module involve setting up the environment for the language processing program it implements, or "wraps" (e.g., processing arguments given by the user via the GGI), as well as retrieving information from the GDM, invoking the program, and taking care of the output produced, that is, format it and record it in the GDM.

2.3 CREOLE modules in SVENSK

From GATE's point of view, SVENSK is a set of CREOLE objects. The language processing software wrapped by the CREOLE objects are in-house modules, commercially available mod-

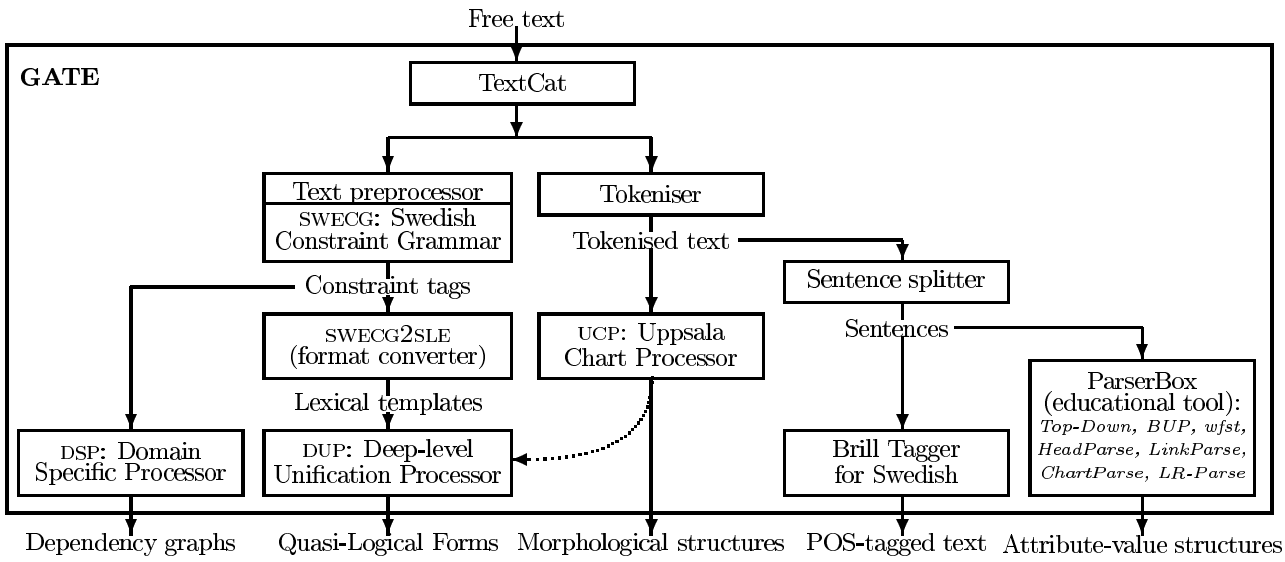

Figure 1: How the modules in SVENSK are interconnected to form different processing chains.

ules, and modules from Swedish academia. The modules integrated so far are shown in Figure 1.

As indicated in the figure, there are different ways the input texts can take through the system. At the top end of the picture, van Noord's freely available[1] language identifier TextCat constitutes the starting point for all processing chains in SVENSK. Here, it allows the user to restrain the input to the system to be in Swedish.

We then have two main options, either to pass the input through SWECG, the Swedish version of LingSoft oy's and Helsinki University's Constraint Grammar (Karlsson et al., 1995), or through a parallel sequence of tokenisation and sentence segmentation developed specifically for the SVENSK project (Olsson, 1998).

The processing chains then split further, and — from left to right in Figure 1 — end in the following modules;

- DSP, the Domain Specific Processor (Sunnehall, 1996) produces shallow dependency graphs intended for use in applications requiring a robust interface for a specific application, such as the Olga dialogue system (Beskow et al., 1997);
- DUP, the Deep-level Unification-based Processor, a component made up of a large-scale unification-based grammar for Swedish (Gambäck, 1997) and an LR-parser (Samuelsson, 1994). It yields a relatively 'deep' level of analysis but at the cost of robustness, and has previously been used for machine translation and database interfacing projects, including the SICS-SRI-Telia "Spoken Language Translator" (Rayner et al., 1993).
- the Uppsala Chart Processor (Sågvall-Hein, 1981) produces morphological analyses;
- a Swedish version (Prütz, 1997) of the Brill Tagger (Brill, 1992);
- the ParserBox, an educational tool consisting of seven parsers operating on a small grammar. At this end the different ways the parsers process the input is of main interest, rather than the output produced.

Each of the components has a standardised input/output interface, users will have the choice of working with the supplied development system — as may be appropriate for academic research on particular aspects of language use — or selecting and combining modules for integration into a user application. Since the I/O interfaces conform to the annotation model of the TIPSTER architecture (Grishman, 1995) developers (and users) can easily add components to the platform, and then link them together to form an application.

[1] At `www.let.rug.nl/~vannoord/TextCat`

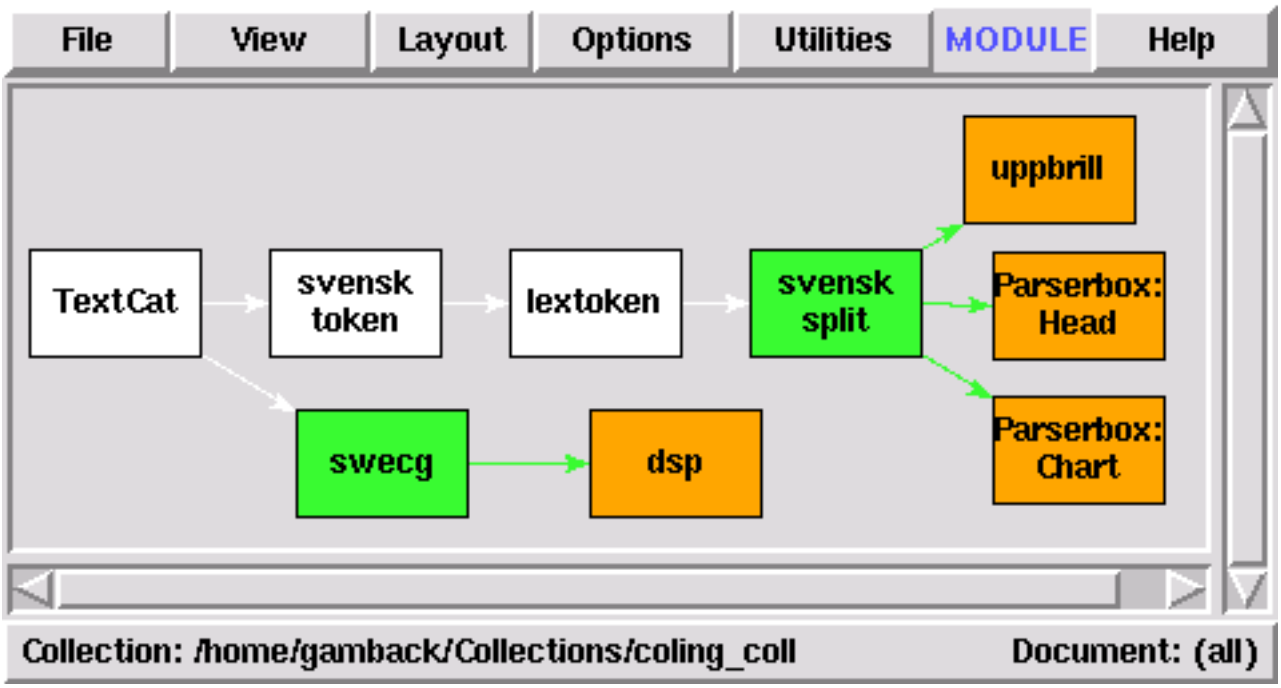

Figure 2: An example of a system built by some of the SVENSK modules.

Alternatively, two components with the same interfaces and functionality can be defined for the platform, and then evaluated in the same application. This allows students to experiment with different approaches to a linguistic problem (such as parsing, using the algorithms supplied on the ParserBox), or research experiments to use the most appropriate component for their purposes and performance criteria (such as speed, robustness, etc).

Figure 2 shows how some of the SVENSK modules can be linked within GATE to form four different processing chains, in this case with three possible output levels: dependency-based semantics (from DSP), POS-tagged text (from Brill), or syntactic analyses obtained from either a chart parser or a head-based parsing strategy.

3 Related work

Over the years there have been many efforts in the direction of creating large toolsets for language processing. Some have been built with one particular application — or class of applications — in mind, but mostly the more or less explicit aim has been to create reusable toolsets for a wide range of tasks. In this section we will look at some of the major stepping stones.

3.1 Setback 1: Eurotra

Back in 1977, the first steps were taken towards what would become the most ambitious effort in the field seen so far. The goal of the Eurotra Programme was to develop a machine translation system collaboratively in all the member-states of the (then) European Community. A working group tried to establish linguistic and software standards as the basis for the project, but the amount of work done in this group was intended to be small, while the main efforts were to be localised to centra in the different states, working on some of the (at the end) nine languages and 72 language pairs (King and Perschke, 1987; Bech and Nygaard, 1988).

After the project, "Eurotra bashing" has developed into something of a sport for European computational linguists, resulting in that the reasons for *why* the project failed (at least partially[2]) have in themselves not been dis-

[2]'Failure' is a relative notion, since it was not originally a goal of the Eurotra Programme to build one production-quality system. The degrees of freedom left for the different groups had the positive effect of building up language processing competence and infrastructure in the EC countries and of producing some working, full-scale "spin-off" systems, such as PaTrans (Hansen, 1994).

cussed enough. Certainly, the lack of overall coordination soon became a liability. Inherent short-comings of the formalisms, and inefficiency of the implementation related to fundamental problems with the formalisms are another reason which have been pointed out (Crookston, 1990; Pulman et al., 1991).

However, the main problem with the framework was probably that it never in itself moved towards one system; indeed, Johnson and Rosner (1987) discussed a software environment for Eurotra building on tools for rapid implementation and evaluation of a variety of experimental theories. In the spirit of that several parallel systems and formalisms were used, and the formalisms changed rapidly over time. Still, no framework was developed which could accommodate these different types of modules, no clear interfaces were designed, no central instance under-took the integration task.

3.2 Success Story 1: CLE

In contrast to the multi-site Eurotra effort, SRI International's Cambridge Research Centre and Cambridge University's Computer Laboratory in 1985 suggested a UK-internal project developing a Core Language Engine (CLE), a domain independent system for translating English sentences into formal representations (Alshawi et al., 1992). SRI's CLE built on a modular-staged design in which explicit intermediate levels of linguistic representation were used as an interface between successive phases of analysis.

The CLE has been applied to a range of tasks, including machine translation and interfacing to a reasoning engine. The modular design also proved well suited for porting to other languages and the implementation was quite efficient. Thus, the project proved its purpose. However, even though the CLE system received considerable attention, it failed to spread in the community, the main reason being that it simply was too expensive to obtain it.[3]

3.3 Setback 2: ALEP

Following the Eurotra tradition, ALEP, the Advanced Language Engineering Platform (Simpkins and Groenendijk, 1994; Bredenkamp et al., 1997), introduced in Section 2.1, was another European Commission initiative to provide the European language research and engineering community with a general purpose research and development environment.

The ALEP platform supplied a range of processing resources and was particularly targeted at supporting multilinguality. However, it imposed its own formalisms (for grammars, etc.) on the developers and users. In addition, the initial implementations were as inefficient as Eurotra's and ALEP never became widely spread.

3.4 Success Story 2: Verbmobil

The real contrast to Eurotra came with Verbmobil, a multi-site German government effort which started in the early 90's (Kay et al., 1994). The main processing stream of the project was clearly defined (even though parallel and complementary modules were allowed) and the interfaces between different groups and modules were developed during the project in intense inter-group discussions.

A major reason why the project succeeded in producing an overall joint system was a concentrated effort by a central system administration group which incorporated components developed at several different sites and in many different programming paradigms into one platform (Bub and Schwinn, 1996). The Verbmobil architecture employs ICE, Intarc Communication Environment (Amtrup, 1997), a general communication package, but which of course has been primarily developed for the specific needs of the Verbmobil task, non-incremental multilingual spoken dialogue translation.

3.5 Success Story 3: GATE

In the mid 90's, the UK Engineering and Physical Sciences Research Council (EPSRC) started to fund a project at the University of Sheffield aimed at building a General Architecture for Text Engineering, GATE (Cunningham et al., 1996; Gaizauskas et al., 1996; Cunningham et al., 1997; Cunningham et al., 1999).

As described in Section 2.2, GATE does not adhere to a particular linguistic theory, but is rather an architecture and a development environment designed to fit the needs of researchers and application developers. It presents users with an environment in which it is easy to use and integrate tools and databases, all accessi-

[3] "You don't give away a one million pound program" (SRI research manager). Contrast this with the strategy of, e.g., John McAfee to give away his antivirus software for free — and making millions on selling the updates!

Table 1: Language processing resources in SVENSK

Processing resource	Main task	Author(s)
TextCat	Language identification	[van Noord; U Groningen]
Tokeniser	Tokenisation	[Olsson; SICS & Uppsala U]
LexToken	Lexicalised phrase tokenisation	[Hassel, Johansson; SICS & Sthlm U]
Sentence splitter	Segmentation	[Olsson; SICS & Uppsala U]
Swedish Brill Tagger	Part-of-speech tagging	[Prütz; Uppsala U]
Uppsala Chart Processor	Morphology	[Sågvall-Hein; Uppsala U]
LP-Detect	Lexicalised phrase recognition	[J. Lindberg; Stockholm U]
Swedish Constraint Grammar	Morphosyntactic analysis	[LingSoft oy & Helsinki U]
SWECG2CLE	Format converter	[Eriksson; SICS]
Deep-level Unification-based Processor	Swedish grammar & LR-parser	[Gambäck; SICS], [Samuelsson; SICS]
Domain-specific Processor	Dependency structure semantics	[Sunnehall; SICS]
ParserBox	Educational tool	[Eineborg, Olsson et al; SICS]

ble through a friendly user interface. The platform is free for non-commercial and research purposes, and has so far been distributed to more than 250 different sites around the world.[4]

3.6 Meanwhile in the US... Galaxy

In the US, there have also been some efforts in the direction of open architectures that incorporates language processing resources, in particular within the research programmes sponsored by DARPA, the Defense Advanced Research Projects Agency. With TIPSTER (Grishman, 1995), the design of a general architecture was agreed upon; however, the full TIPSTER annotation scheme (Grishman and others, 1997) has not been implemented as such. Instead, the MITRE Cooperation is currently (under DARPA funding) developing Communicator, a testbed similar to the Verbmobil one.

The initial DARPA Communicator architecture builds on MIT's Galaxy system (Seneff et al., 1998). A central process, the Hub, is connected with a variety of server processes and governs the control flow between them. A wide range of component types are supported: language understanding and generation, speech recognition and synthesis, dialogue management, and context tracking (Goldschen and Loehr, 1999).

The goal of the Communicator — to provide an architecture used by everyone, easing the work of porting modules and system evaluation — seems decent in itself; however, the system has not been that well received within the US research community. ("We've spent most of our time the last year trying to make our stuff follow Communicator standards, rather than on doing research," anonymous US researcher, personal communication 2000).

4 Issues in composing a toolset

A result of the integration in SVENSK is that programs from many different sources and backgrounds, which originally were not built to communicate which each other are doing this now. Table 1 shows the main tasks of all the SVENSK modules, as well as the author(s) and sources behind the different units.

Collecting and distributing algorithmic resources and making different programs interoperate present a wide range of challenges, along several different dimensions; we will denote the key dimensions 'diplomatic', technical, and linguistic. In the rest of this section we will discuss some of the experiences we have drawn from the project with regards to these dimensions.

[4] In June 2000, according to the "incomplete" list of licensees given on the GATE web pages: `www.dcs.shef.ac.uk/research/groups/nlp/gate`

4.1 Diplomatic challenges

With 'diplomatic', we mean some of the conclusions which can be drawn from the examples of other systems in Section 3. Eurotra and ALEP both had the problem of linguists not wanting to agree on formalism standards while a framework supporting diversity was lacking. The CLE was successful as a system, but commercial shortsightedness destroyed its chances of more widespread popularity.

Commercial interests have also been a problem within SVENSK, but we have also seen that it is hard to get access to academic LE resources. The need for component reuse is often appreciated by everybody in the field. However, to put action behind words is not as easy. In particular, researches need to be convinced to invest the extra time and resources to package their components in an exportable and reusable form.

Table 2: Resource availability

Availability	Resource(s)
In-house and free	DSP, DUP, Tokeniser, LexToken, ParserBox, Sentence splitter
External and free	TextCat, GATE, LP-Detect
External, restricted	UCP, Swedish Brill
Commercial, closed	SWECG

A key aim of the project has been that the resources included in SVENSK should be freely available for non-commercial use, at least for Swedish institutions. As can been seen in Table 2 all current components except for SWECG meet this requirement.[5] Of course, processing resources included in the system in the future should preferably also match this free-for-all strategy.

Still, making language processing resources freely available and, in particular, reusability of resources is really a very uncommon concept in the computational linguistic community. Possibly this also reflects another uncommon concept, that of experiment reproducibility; in most research areas the possibility for other researchers to reproduce an experiment is taken for granted. Yes, this is the very core of what is accepted as good research at all. Strangely enough, this is rarely the case in Computer Science in general and definitely not within Computational Linguistics.

We believe that this will change and that reproducibility will be generally accepted as a criteria of good research even in Computational Linguistics. And to give other researchers the option of reproducing an experiment means giving them access to the language engineering resources used in the experiment. Convincing the members of the CL research community to both make their own processing resources freely available to the rest of the community and actually even to try to reuse somebody else's resources is indeed a tough 'diplomatic' challenge.

4.2 Technical/software challenges

From the technical point of view, one major conclusion is that the difficulties of integrating language processing software never can be overestimated. Even when using a liberal framework like GATE it is hard work making different pieces of software from different sources and built according to different programming traditions meet any kind of interface standard.

To give the flavour of the problem, Table 3 singles out the underlying implementation languages of some of the SVENSK components, while diversity in software authors and sources was shown already in Table 1.

Table 3: Implementation languages

Language	Resource(s)
Prolog	DSP, DUP, ParserBox
C/C++	SWECG, Brill, Tokeniser, GATE (part)
Tcl/Tk	GATE
Perl	TextCat, Brill (part), LexToken, LP-Detect, Sentence splitter
LISP	UCP

The application programming interface thus moves to the centre of attention: No matter how

[5] SWECG, the Swedish Constraint Grammar, is available from LingSoft oy, Helsinki, unfortunately for a currently discouragingly high license fee, albeit reduced for academic SVENSK users.

linguistically adequate a piece of language processing software is, without a proper API it cannot be used in conjunction with other programs.

In a way, it is understandable that academia does not always put much effort in packaging and documenting their software, since their main purpose is not to sell and widely distribute it. The trouble is that some of the actors on the commercial scene do not document their systems in a proper manner, either. Far too often this has resulted in inconsistencies with the input and output of other modules. This probably reflects a certain level of immaturity in the field when it comes to software development; the problem might solve itself when the level of competition between different companies increases.

Software portability is another issue: The components available often rely on a particular operating system or a particular software environment to work, something which may cause problems in settings when you wish to distribute your collected efforts to other parties, or when you wish to add a new component to your collection. If you are not careful when integrating components that are not first and foremost intended to function together, it is not likely that their combined performance will even level with the performance of the individual programs regarding, e.g., time and memory requirements.

In software industry in general, it is hard to recreate the situations where bugs occur, and it is even harder to correct them once you have found them. When collecting and integrating a set of heterogenous language processing components, the problem of localising a bug is even harder. As long as the source code of the software under consideration is available, you might be able to correct the bugs yourself; however, software delivered as a "black-box" (that is, if it is impossible to access the code inside, which is common for commercial software), allow no one to remedy even the smallest flaw.

4.3 Linguistic challenges

Of course, LE components differ with respect to such things as language coverage, processing accuracy and the types of tasks addressed. It is also the case that tasks can be done at various levels of proficiency. The trouble is that there is no quality control available to either the toolbox developer nor to the end-user. If a large number of LE components are to be integrated, they should first be categorised so that components with a great difference in, say, lexical coverage are not combined.

A familiar problem for all builders of language processing systems relates to the adaptation to new domains. When reusing resources built by others this becomes even more accentuated, especially if an LE resource is available only in the "black box" form (and thus relates to the issues of the previous subsection).

In general, the power required of a language processing system is affected by three main factors: the type of task involved, the needs of the specific application, and the application domain, including the vocabulary and the register (sublanguage) complexity. It seems impossible — or at least *very* hard! — to compose a really general toolkit. Toolkits will always have to focus on some classes of tasks and applications and/or on some language and operation domains.

A classification of the SVENSK components according to three different limitation dimensions is given in Table 4.

Table 4: Linguistic limitations

Limitations	Resource(s)
Language dependent	All except TextCat and Sentence splitter (some)
Domain dependent	DSP (most others more or less)
Sentence-based	DSP, DUP, ParserBox

5 Conclusions

- A toolset should not be *too* general. There has to be some focus on its end-usage, at least to some manageable set of classes of tasks and applications.
- The portability issues across operating systems as well as institutional borders depends on technical issues such as licenses and availability. The "*a chain is never stronger than its weakest link*"-metaphor is certainly applicable!
- The domain and coverage of language processing software is an additional obstacle; it is important to match pieces of software accordingly!

Acknowledgements

The SVENSK project has been funded by SICS and NUTEK under grants P5475 and P13338-1.

Several persons have been involved in the project since its start in 1995, in particular Mikael Eriksson who did the first work on the GATE interfacing. Mats Wirén and Barbro Atlestam were the key persons behind ensuring the project's funding. Scott McGlashan, Charlotta Berglund, Victoria Johansson and Kristina Hassel all worked directly on SVENSK at SICS, while Christer Samuelsson, Jussi Karlgren, Nikolaj Lindberg, Lena Santamarta and Ivan Bretan worked on other SICS projects which contributed indirectly to the system.

Klas Prütz, Anna Sågvall-Hein and Jan "Beb" Lindberg donated language processing resources to the project. Gertjan van Noord sat an example for all researchers by making his software freely available, as did Hamish Cunningham and the rest of the Sheffield GATE group. Thanks also to the members of SVENSK's scientific advisory board which have greatly influenced the project: Barbro, Anna, Mats, Robin Cooper, Lars Ahrenberg, and Calle Welin.

References

Hiyan Alshawi, editor, David Carter, Jan van Eijck, Björn Gambäck, Robert C. Moore, Douglas B. Moran, Fernando C. N. Pereira, Stephen G. Pulman, Manny Rayner, and Arnold G. Smith. 1992. *The Core Language Engine*. MIT Press, Cambridge, Mass.

Jan W. Amtrup. 1997. ICE: A communication environment for Natural Language Processing. In *Proc. International Conference on Parallel and Distributed Processing Techniques and Applications*, Las Vegas, Nevada.

Annelise Bech and Anders Nygaard. 1988. The E-framework: A formalism for Natural Language Processing. In *Proc. 12th International Conference on Computational Linguistics*, volume 1, pages 36–45, Budapest, Hungary. ACL.

Jonas Beskow, Kjell Elenius, and Scott MacGlashan. 1997. Olga — a dialogue system with an animated talking agent. In G. Kokkinakis, N. Fakotakis, and E. Dermatas, editors, *Proc. 5th European Conference on Speech Communication and Technology*, volume 3, pages 1651–1654, Rhodes, Greece. ESCA.

Andrew Bredenkamp, Thierry Declerck, Frederik Fouvry, Bradley Music, and Axel Theofilidis. 1997. Linguistic engineering using ALEP. In (Mitkov and Nicolov, 1997), pages 92–97.

Eric Brill. 1992. A simple rule-based part of speech tagger. In *Proc. 3rd Conference on Applied Natural Language Processing*, pages 152–155, Trento, Italy. ACL.

Thomas Bub and Johannes Schwinn. 1996. Verbmobil: The evolution of a complex large speech-to-speech translation system. In *Proc. 4th International Conference on Spoken Language Processing*, Philadelphia, Pennsylvania.

Ian Crookston. 1990. The E-framework: Emerging problems. In H. Karlgren, editor, *Proc. 13th International Conference on Computational Linguistics*, volume 2, pages 66–71, Helsinki, Finland. ACL.

Hamish Cunningham, Yorick Wilks, and Robert J. Gaizauskas. 1996. GATE — a General Architecture for Text Engineering. In *Proc. 16th International Conference on Computational Linguistics*, volume 2, pages 1057–1060, København, Denmark. ACL.

Hamish Cunningham, Kevin Humphreys, Robert J. Gaizauskas, and Yorick Wilks. 1997. Software infrastructure for Natural Language Processing. In *Proc. 5th Conference on Applied Natural Language Processing*, Washington, DC. ACL.

Hamish Cunningham, Robert J. Gaizauskas, Kevin Humphreys, and Yorick Wilks. 1999. Experience with a Language Engineering architecture: Three years of GATE. In *Proc. Workshop on Reference Architectures and Data Standards for NLP*, Edinburgh, Scotland. AISB.

Mikael Eriksson and Björn Gambäck. 1997. SVENSK: A toolbox of Swedish language processing resources. In (Mitkov and Nicolov, 1997), pages 336–341.

Robert Gaizauskas, Hamish Cunningham, Yorick Wilks, Peter Rodgers, and Kevin Humphreys. 1996. GATE: An environment to support research and development in Natural Language Engineering. In *Proc. 8th International Conference on Tools with AI*, Toulouse, France. IEEE.

Björn Gambäck. 1997. *Processing Swedish Sentences: A Unification-Based Grammar and*

some Applications. PhD Thesis, The Royal Institute of Technology, Dept. of Computer and Systems Sciences, Stockholm, Sweden.

Björn Gambäck and Fredrik Olsson. 2000. Experiences of Language Engineering algorithm reuse. In *Proc. 2nd International Conference on Language Resources and Evaluation*, volume 1, pages 161–166, Athens, Greece. ELRA.

Alan Goldschen and Dan Loehr. 1999. The role of the DARPA Communicator architecture as a human computer interface for distributed simulations. In *Spring Simulation Interoperability Workshop*, Orlando, Florida. SISO.

Ralph Grishman et al., 1997. *TIPSTER Text Phase II Architecture Design. Version 2.3.* New York, NY.

Ralph Grishman, 1995. *TIPSTER Phase II Architecture Design Document (Tinman Architecture) Version 1.52.* New York, NY.

Viggo Hansen. 1994. PaTrans — a MT-system: Development and implementation of and experiences from a MT-system. In *Proc. 1st Conference of the Association for Machine Translation in the Americas*, pages 114–121, Columbia, Maryland. AMTA.

Rod Johnson and Mike Rosner. 1987. Machine translation and software tools. In (King, 1987), chapter 11, pages 154–167.

Fred Karlsson, Atro Voutilainen, Juha Heikkilä, and Arto Anttila, editors. 1995. *Constraint Grammar: A Language-Independent System for Parsing Unrestricted Text.* Mouton de Gruyter, Berlin, Germany.

Martin Kay, Jean Mark Gawron, and Peter Norvig. 1994. *Verbmobil: A Translation System for Face-to-Face Dialog.* Number 33 in Lecture Notes. CSLI, Stanford, California.

Maggie King, editor. 1987. *Machine Translation Today: the State of the Art.* Edinburgh University Press, Edinburgh, Scotland.

Maggie King and Sergei Perschke. 1987. EUROTRA. In (King, 1987), chapter 19, pages 373–391.

Ruslan Mitkov and Nicolas Nicolov, editors. 1997. *Proc. 2nd International Conference on Recent Advances in Natural Language Processing*, Tzigov Chark, Bulgaria.

Fredrik Olsson. 1998. Tagging and morphological processing in the SVENSK system. MA Thesis, Uppsala University, Sweden.

Fredrik Olsson, Björn Gambäck, and Mikael Eriksson. 1998. Reusing Swedish language processing resources in SVENSK. In *Proc. Workshop on Minimizing the Effort for Language Resource Acquisition (LREC98)*, pages 27–33, Granada, Spain. ELRA.

Klas Prütz. 1997. Preparing a training corpus in Swedish for training an automatic part of speech tagging system. In H. Kalverkämper and B. Svane, editors, *Übersetzen und Dolmetschen. Forschungsstand und Perspektive. Translation and Interpreting. State and Perspectives. Proc. Humboldt-Stockholm Symposium*, Stockholm University, Sweden.

S. G. Pulman, editor, H. Alshawi, D. J. Arnold, D. M. Carter, J. Lindop, K. Netter, J. Tsujii, and H. Uszkoreit. 1991. *Eurotra ET6/1: Rule Formalism and Virtual Machine Design Study.* CEC, Luxembourg.

Manny Rayner, Ivan Bretan, David Carter, Michael Collins, Vassilios Digalakis, Björn Gambäck, Jaan Kaja, Jussi Karlgren, Bertil Lyberg, Steve Pulman, Patti Price, and Christer Samuelsson. 1993. Spoken language translation with mid-90's technology: A case study. In *Proc. 3rd European Conference on Speech Communication and Technology*, volume 2, pages 1299–1302, Berlin, Germany. ESCA.

Anna Sågvall-Hein. 1981. An overview of the Uppsala Chart Parser version 1 (UCP-1). Technical report, Center for Computational Linguistics, Uppsala University, Sweden.

Christer Samuelsson. 1994. *Fast Natural-Language Parsing Using Explanation-Based Learning.* PhD Thesis, The Royal Institute of Technology, Dept. of Computer and Systems Sciences, Stockholm, Sweden.

Stephanieo Seneff, Ed Hurley, Raymond Lau, Christine Pao, Philipp Schmid, and Victor Zue. 1998. Galaxy–II: A reference architecture for conversational system development. In *Proc. 5th International Conference on Spoken Language Processing*, volume 3, pages 931–934, Sydney, Australia.

Neil Simpkins and Marius Groenendijk. 1994. The ALEP project. Technical report, Cray (now Anite) Systems / CEC, Luxembourg.

Joel Sunnehall. 1996. Robust parsing using dependency with constraints and preference. MA Thesis, Uppsala University, Sweden.