Experiences of Language Engineering Algorithm Reuse

Björn Gambäck, Fredrik Olsson

Information and Language Engineering Group
Swedish Institute of Computer Science
Box 1263, S–164 29 Kista, Sweden
{gamback, fredriko}@sics.se

Abstract

Traditionally, the level of reusability of language processing resources within the research community has been very low. Most of the recycling of linguistic resources has been concerned with reuse of data, e.g., corpora, lexica, and grammars, while the algorithmic resources far too seldom have been shared between different projects and institutions. As a consequence, researchers who are willing to reuse somebody else's processing components have been forced to invest major efforts into issues of integration, inter-process communication, and interface design.

In this paper, we discuss the experiences drawn from the SVENSK project regarding the issues on reusability of language engineering software as well as some of the challenges for the research community which are prompted by them. Their main characteristics can be laid out along three dimensions; technical/software challenges, linguistic challenges, and 'political' challenges. In the end, the unavoidable conclusion is that it definitely is time to bring more aspects of engineering into the Computational Linguistic community!

1. Introduction

It is a generally accepted belief that data can be reused, and there are today several repositories which collect and distribute data; however, there have so far not been many attempts to do the same for the programs which operate on these data. There are a few frameworks and platforms available which aim to ease program and algorithm reuse, but the distribution and collection of algorithmic resources is more or less non-existent. Possibly because the programs tend to be more specific in terms of, e.g., platform, time, and space requirements than the data are.

The SVENSK project at SICS, the Swedish Institute of Computer Science, is an attempt to collect Swedish language processing resources and to integrate them in one common framework, GATE, General Architecture for Text Engineering from the University of Sheffield. The main point is that by collecting a large set of resources under one roof it will be easier for researches and non-commercial organisations in Sweden to get a grasp of what type of resources actually are available, to reuse the ones which fit their needs, and to distribute the results of their own research to others.

The rest of the paper discusses the experiences we have drawn from the project and some of the challenges for the research community which follow from them. However, we start out with a short overview of other previous and present efforts along similar lines, as well as of the SVENSK project as such.

2. Algorithm reuse vs. data reuse

Traditionally, the level of reusability of language processing resources within the research community has been very low. Most of the recycling of linguistic resources has been concerned with reuse of data, that is, of corpora, lexica, and (to some extent) grammars, while the algorithmic resources far too seldom have been shared between different projects and institutions.

Even though there are some repositories for algorithms, e.g., the DFKI Natural Language Software Registry (an initiative of the Association for Computational Linguistics) in the same fashion as there are archives for data (such as the Linguistic Data Consortium or the European Language Resources Association), a prime obstacle to reuse of algorithmic resources has been that there have been few easily-available development platforms for language engineering (LE). A researcher willing to reuse somebody else's processing components has been forced to invest major efforts into issues of integration, inter-process communication, and interface design.

In Europe, there have so far been few efforts to design and build general purpose LE platforms. ALEP, the Advanced Language Engineering Platform (Simpkins and Groenendijk, 1994; Bredenkamp et al., 1997) was an initiative of the European Commission which aimed in this direction. It provided a range of processing resources and was particularly targeted at supporting multilinguality. ALEP did, however, impose its own formalisms (for grammars, etc.) on the developers and users. In addition, ALEP was very slow (Eriksson and Gambäck, 1997) and the system never became widely spread.

Another relevant platform, although directed towards one particular application, is the German Verbmobil architecture (Bub and Schwinn, 1996), which incorporates components developed at several different sites and in many different programming paradigms. It employs a communication package called ICE (Intarc Communication Environment), but is of course primarily developed for the specific needs of the Verbmobil project, a multilingual speech-to-speech dialogue translation task.

In the US, there have been some efforts in the direction of open architectures that incorporates language processing, both by single companies — such as the multimodal Open Agent Architecture™ from SRI International (Cohen et al., 1994; Moran et al., 1997) — and in particular within the research programmes sponsored by DARPA, the Defense Advanced Research Projects Agency, where a design of a general architecture called TIPSTER (Grishman, 1995) was agreed upon. However, the full TIPSTER annotation scheme (Grishman and others, 1997) has not been implemented as such.

In a recent DARPA funded programme called Communicator, the MITRE Cooperation is developing a testbed similar to the Verbmobil one. The initial DARPA Communicator architecture (DCA) extends the MIT Galaxy system (Seneff et al., 1998) and provides an environment for testing a wide range of types of components: language understanding and generation, speech recognition and synthesis, dialogue management, and context tracking (Goldschen and Loehr, 1999). In the DCA a central process called the Hub is connected with a variety of server processes and controls the control flow between them.

Going back to Europe again, finally, GATE, the General Architecture for Text Engineering, is an open platform for LE developed by the University of Sheffield, UK. As it turns out, GATE (as described in the next section) is the most suitable platform for our needs in developing and compiling a collection of LE software in the SVENSK project.

3. GATE

The GATE language engineering platform (Cunningham et al., 1996) is developed at the University of Sheffield and funded by the U.K. Engineering and Physical Sciences Research Council (EPSRC). GATE provides a communication and control infrastructure for linking together language engineering software. It 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. GATE is available free of charge for non-commercial use, and, for demonstration purposes, it comes with a set of components that form a system called VIE, Vanilla Information Extraction (Humphreys et al., 1996).

GATE supports reuse of resources, data as well as algorithms, since it provides for well-defined, user-friendly application programmers interfaces (APIs). Once a module has been integrated in the system, it is very easy to combine it with already existing modules to form new systems. GATE contributes to the portability of components in the sense that software written in programming languages stemming from completely different paradigms can be mixed.

Each component integrated into GATE has a standard I/O interface, which conforms to a subset of the TIPSTER annotation model. This means that GATE compatibility equals TIPSTER compatibility, allowing developers (and users) to easily add all types of TIPSTER compatible components to the platform, and then link them together to form an application.

The infrastructure of GATE provides several levels of integration, reflecting how closely a new module should be connected to the core system; however, at all integration levels a wrapper code must surround the core module code. This wrapper describes the communication between the module and the GATE system, i.e., the mapping between the I/O of the module and the corresponding TIPSTER primitives as implemented in the GATE. Also, for each module, there must be a specification of the type of input and output data of the module in terms of the TIPSTER database annotation scheme. This is the information that GATE needs in order to connect the module to the other integrated modules in the system.

4. The SVENSK project

The SVENSK project (Eriksson and Gambäck, 1997; Olsson et al., 1998) is a national effort funded by the Swedish National Board for Industrial and Technical Development (Nutek) and SICS addressing the problem of reusing LE software. The aim has been to develop a multi-purpose language processing system for Swedish based, where possible, on existing components. Rather than building a monolithic system attempting to meet the needs of both academia and industry, the project has created a general toolbox of reusable language processing components and resources, primarily targeted at teaching and research. The SVENSK system as such is mainly the sum of a fairly large set of different reusable language resources.

The reusability of the LE components in SVENSK system arises from having each component integrated into GATE. SICS has so far incorporated a wide range of different modules: in-house modules, commercially available modules, modules freely available on the internet, and modules from Swedish academia. The main characteristica of the modules are outlined in the next section.

The SVENSK project was divided into three phases, the first of which was covering the period from spring 1996 to the end of 1996 and the second from the beginning of 1997 to August 1997. The third phase started in January 1998 and ran until the end of 1999. In this sense, the project as such has been concluded and below we will discuss our experiences from it. However, as a national repository for language processing resources, the SVENSK system will live on. SICS has thus obtained some continued funding for the system for distributing it and for supporting Swedish academia in the process of incorporating new language processing resources into it.

4.1. Language Components in SVENSK

The modules which so far have been included in SVENSK are shown in Figure 1. The components stem from a wide range of paradigms and were developed in several projects at different institutions. Thus, they

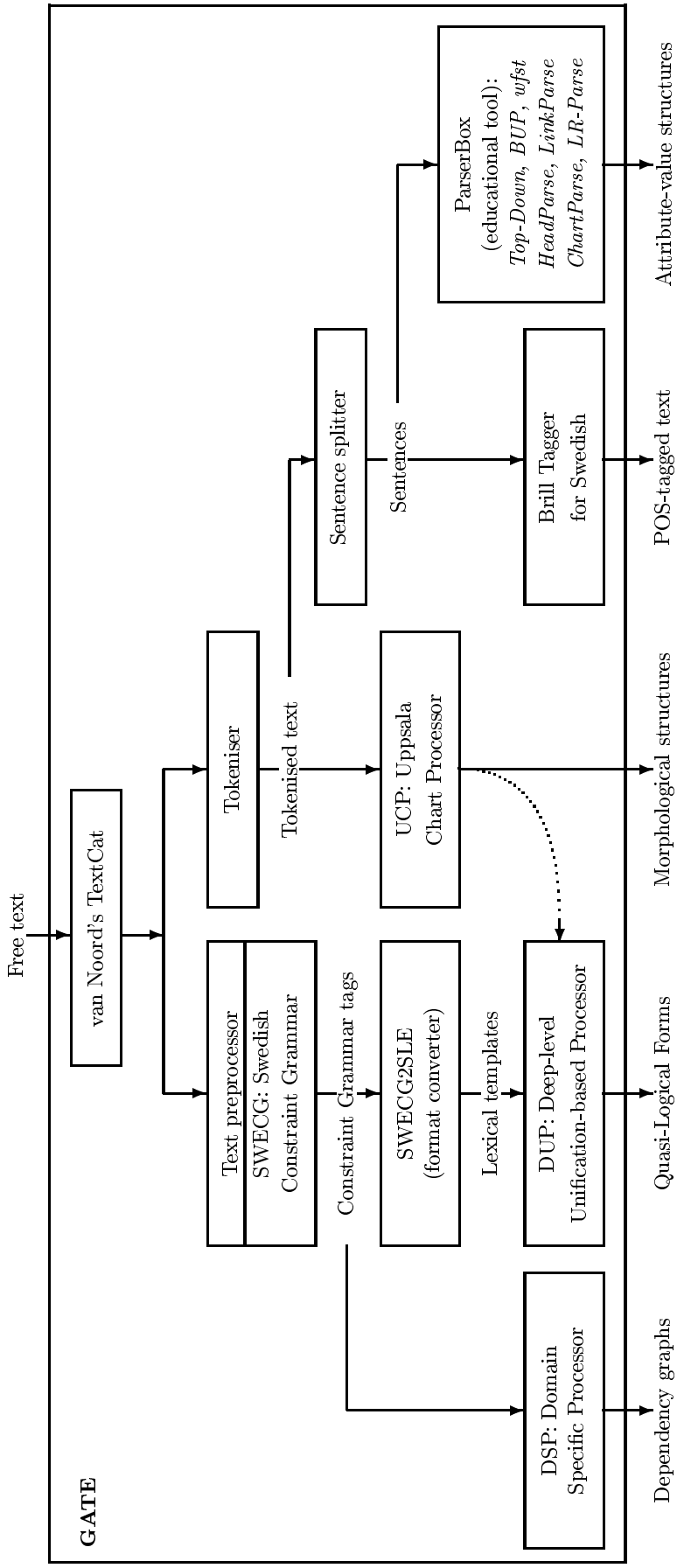

Figure 1: How the modules in SVENSK are interconnected to form different processing chains.

were not primarily designed and implemented with their interoperability in mind.

Two of the modules are commercially available programs for morphological and functional analysis, both from LingSoft OY, Helsinki. Another component, van Noord's text categoriser and language identifier is freely available on the Internet.[1] Swedish academia have contributed a version of Brill's PoS tagger and the morphological processing of the Uppsala Chart Processor. Some other components were previously developed at SICS: a deep-level unification-based processor, a domain-specific processor, the "parser-box" (a toolkit for educational purposes including several types of parsers), as well as segmentation and tokenisation modules.

The input to SVENSK is plain text files, converted by GATE to an internal format based on the TIPSTER annotation scheme. There are several possible processing chains in SVENSK. Currently all start with van Noord's TextCat which is intended as an optional filter for avoiding processing texts in other languages than Swedish (one could, for instance, easily add a module prior to TextCat which, given a set of URLs, fetches texts from the Internet and forwards them to TextCat, which then decides which texts should be further processed by the system).

Starting with the left-most processing chain shown in Figure 1, the output from SVENSK, is as follows:

- the Domain Specific Processor (Sunnehall, 1996) produces shallow dependency intended for use in applications that require a robust interface for a specific application, such as the Olga dialogue system (Beskow et al., 1997);

- the second chain ends in a deep-level unification-based processor, a component made up of a fairly large-scale grammar for Swedish (Gambäck, 1997) and an LR-parser operating on the unification-based grammar formalism (Samuelsson, 1994). The grammar has been used for machine translation and database interfacing projects and yields a relatively 'deep' level of analysis but at the cost of robustness;

- the Uppsala Chart Processor (Sågvall-Hein, 1981) finishes off the third chain, which produces morphological analyses of the input text;

- the last but one processing chain results in part-of-speech tagged text produced by a Swedish version (Prütz, 1997) of the Brill Tagger (Brill, 1992);

- finally, the last chain ends in ParserBox which is an educational tool consisting of seven parsers which operate on a small grammar. The output at this end is not as interesting as are the different ways the parsers process the input.

[1] At `http://odur.let.rug.nl/~vannoord/TextCat/`

5. Experiences

A result of the integration in SVENSK is that programs which originally were not built to communicate which each other are doing this now. Moreover, collecting and distributing algorithmic resources and making different programs interoperate present a wide range of challenges, along several different dimensions, both technical, linguistic, and 'political'. We will discuss the experiences we have drawn from the project and some of these challenges in this section.

5.1. Technical/software challenges

One of the conclusions of the project is that the difficulties of software integration never can be overestimated. Even when using a fairly liberal framework like GATE, but with predefined interface standards, it is hard work making different components from different sources and built according to different programming traditions meet *any* kind of interface standard. Far too often developers of language engineering components do not put enough effort in designing and defining the API. No matter how linguistically adequate a piece of LE software is, without a proper API it is hard to use it in conjunction with other programs.

In addition, just because a set of language engineering components are able to communicate, the performance regarding, e.g., time and memory requirements, does not necessarily improve or even level with the performance of the individual programs. It is important to keep such issues in mind when choosing which components to integrate. In particular when using a platform like GATE which does not impose any "natural" constraints on the components; when language processing resources are designed and implemented for other purposes than that of being integrated in a general tool-box, users are usually aware of the limitations of the components, e.g., by constraints in the user interface implying that a program cannot be used in certain ways. Such limitations are not obvious in an environment like GATE.

The components available are often platform-dependent and difficult to install. The data they produce as well as the type and value ranges of the input data they expect are seldom clearly defined and far too often inconsistent with the input and output data of the other modules. To make things worse, neither the input and output data sets, nor the overall functionality of an LE resource is in general documented. Or, at least, if there is some documentation available, it is commonly either bad, out-dated, or simply faulty. This could certainly be partially remedied if providers of language processing resources supplied some software support, something which academia by tradition back as far away from as possible. This is no surprise, of course. Academic researchers do not (and should not!) work under the programming code quality requirements of publically released, commercial software. And the creators and users of a tool-box such as SVENSK will certainly take this into account. More surprisingly, the commercial companies in the field are

also seriously lacking in both documentation of their products and in software support. This probably reflects a certain level of inmaturity in the field which will solve itself when the level of competition between different companies increases.

5.2. Linguistic challenges

Of course, LE components differ with respect to such things as language coverage and processing accuracy. The trouble is that there is no quality control available to either the tool-box 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 form of a "black box", that is, if it is impossible to access the code inside, as is commonly the case with e.g. commercial systems (and thus relates to the documentation issue of the previous section).

5.3. 'Political' challenges

Another problem has been 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 simple as it may sound. This is not necessarily a consequence of different researchers not really *wanting* other people to use their components and research results, but rather a problem of researches being willing to invest the extra time and resources to package the components in an exportable and reusable form.

Still, reusability of processing 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 not the case in computer science and 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 LE resources used in the experiment. Convincing the CL research community of this is indeed a 'political' challenge.

6. Conclusions

The overall conclusion of the work on collecting and reusing language processing resources is that it is about time that computational linguists, computer scientists and linguists alike start thinking more about how the programs shall be designed and implemented to get better performance and consistency. Still, we do not believe that it is possible to standardise the APIs, other interfaces, annotation schemata, etc., in themselves. However, the basic *principles* which the design of these should follow, ought to be easier to agree on. If other people should have a fair chance of reusing a particular resource, the designer of it should make sure to meet the following requirements:

- document which domain the system is aimed at handling, and the level of success at which it actually does this;
- keep the system architecture open so that other developers have the possibility of modifying it or adding functionality;
- produce tools for testing and evaluation (build test-suites), as well as tools for debugging;
- build a consistent, well-defined, and well-documented API.

The last requirement is probably the most important: It is seldom of interest to others to know exactly what is happening inside a program as long as it is easy to run it, and it is clear what type and range of input data it accepts and expects — and what output data it produces.

Thus, in order to reuse algorithmic resources, we have to:

1. Make sure that researchers share their implementations with others (and how this is to be done is still an open question!).
2. Ask developers to fulfill the requirements above.

This might appear more or less self-evident; however, the fact is that points 1 and 2 very seldom are compatible. At least in the minds of the researchers... It all leaves a lot more to be done for meeting the challenges described in this paper, both the technical and linguistic ones, but maybe in particular the 'political' ones. In total, it definitely is time to bring more aspects of engineering into the Computational Linguistic community!

7. Acknowledgements

SVENSK has been funded by the Swedish National Board for Industrial and Technical Development (Nutek) and SICS. The project has been guided by a reference group composed of academic and industrial members, as well as a Nutek representative: Lars Ahrenberg, Barbro Atlestam, Robin Cooper, Anna Sågvall-Hein, Carl-Wilhelm Welin, and Mats Wirén. Charlotta Berglund, Mikael Eriksson, Scott McGlashan and Joel Sunnehall all contributed to parts of SVENSK, while Ivan Bretan, Jussi Karlgren, and Christer Samuelsson were involved in the initial specifications of the system architecture.

8. References

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, *Proceedings of the 5th European Conference on Speech Communication and Technology*, volume 3, pages 1651–1654, Rhodes, Greece, September. ESCA.

Andrew Bredenkamp, Thierry Declerck, Frederik Fouvry, Bradley Music, and Axel Theofilidis. 1997. Linguistic engineering using ALEP. In RANLP97 (Mitkov and Nicolov, 1997), pages 92–97.

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

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

Philip R. Cohen, Adam J. Cheyer, Michelle Wang, and Soon Choel Baeg. 1994. An open agent architecture. In *AAAI Spring Symposium*, pages 1–8, Stanford University, California, March.

Hamish Cunningham, Yorick Wilks, and Robert J. Gaizauskas. 1996. GATE – a general architecture for text engineering. In *Proceedings of the 16th International Conference on Computational Linguistics*, volume 2, pages 1057–1060, København, Denmark, August. ACL.

Mikael Eriksson and Björn Gambäck. 1997. SVENSK: A toolbox of Swedish language processing resources. In RANLP97 (Mitkov and Nicolov, 1997), pages 336–341.

Björn Gambäck. 1997. *Processing Swedish Sentences: A Unification-Based Grammar and some Applications.* Doctor of Engineering Thesis, The Royal Institute of Technology, Dept. of Computer and Systems Sciences, Stockholm, Sweden, June.

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, March. Simulation Interoperability Standards Organization (SISO).

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

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

Kevin Humphreys, Robert J. Gaizauskas, Hamish Cunningham, and Saliha Azzam, 1996. *VIE Technical Specifications.* Sheffield, England.

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

Douglas B. Moran, Adam J. Cheyer, Luc E. Julia, David L. Martin, and Sangkyu Park. 1997. Multimodal user interfaces in the Open Agent Architecture. In *Proceedings of the International Conference on Intelligent User Interfaces*, pages 61–68, Orlando, Florida, January. ACM.

Fredrik Olsson, Björn Gambäck, and Mikael Eriksson. 1998. Reusing Swedish language processing resources in SVENSK. In *Proceedings of the Workshop on Minimizing the Effort for Language Resource Acquisition*, pages 27–33, at the 1st International Conference on Language Resources and Evaluation, Granada, Spain, May. ELRA.

Klas Prütz. 1997. Sammanställning av en träningskorpus på svenska för träning av ett automatiskt ordklasstaggningssystem. (in Swedish).

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

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

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

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

Joel Sunnehall. 1996. Robust parsing using dependency with constraints and preference. Master of Art Thesis, Uppsala University, Uppsala, Sweden, September.