Building an open-source development infrastructure for language technology projects

Sjur N. Moshagen[1], *Tommi A Pirinen*[2], *Trond Trosterud*[1]

(1) University of Tromsø, Norway
(2) Helsinki university, Finland

`sjur.n.moshagen@uit.no`, `tommi.pirinen@helsinki.fi`, `trond.trosterud@uit.no`

ABSTRACT
The article presents the Giellatekno & Divvun language technology resources, more specifically the effort to utilise open-source tools to improve the build infrastructure, and the solutions to help adapt to best practices for software development. The article especially discusses how the infrastructure has been remade to cope with an increasing number of languages without incurring extra overhead for the maintainers, and at the same time let the linguists concentrate on the linguistic work. Finally, the article discusses how a uniform infrastructure like the one presented can be used to easily compare languages in terms of morphological or computational complexity, coverage or for cross-lingual applications.

KEYWORDS: NoDaLiDa 2013, Infrastructure, Computational linguistics, Finite-state transducers, Language resources, Multilinguality.

1 Introduction

There are about 7000 languages in the world, according to Ethnologue. According to the same source, about 2200 of them have a standardised orthography[1]. Of these, roughly 100-150 have keyboard layouts preinstalled on major operating systems (Trosterud, 2012), and only ∼50 are available as a user interface language. The languages with good, continuous speech recognition can be counted on one hand (domain-specific speech recognition on a couple of hands).

The infrastructure described in this article is targeted at languages with limited textual resources: a small literary body, often no PC keyboard layout standard (and thus not delivered as part of any OS), quite often a limited grammatical description. The goal of the work done for these languages is to develop a grammatical description in the form of a morphological grammar and lexicon, and use the lexicon and morphology as the basis for NLP tools and end user tools such as proofing tools and electronic dictionaries. Most of the targeted languages worked on so far are morphologically rich, and the chosen technology is well suited for this type of languages.

Language technology infrastructure is a fairly broad concept, and is covered in many papers and on-going projects (see a.o. (Broda et al., 2010), (Loper and Bird, 2002)).

Many projects could be seen as *resource infrastructure* projects making resources and tools *available* to developers (e.g. META-SHARE (Federmann et al., 2012), CLARIN (Váradi et al., 2008)), our work could instead be characterised as a *development environment* infrastructure project, where the goal is to improve the language processing. This is close to the objectives of environments such as GATE ((Cunningham et al., 1997), (Cunningham et al., 2011)), NLTK[2] and UIMA/OpenNLP[3]. These are language-independent statistically based toolkits, where the linguistic analysis is left to the annotator. Such tools are less feasible for languages with rich morphology and little text, and they are also not able to do language generation. Another type of language processing tools is HPSG[4] and LFG XLE[5]. These tools are also grammar-based, but their focus is on syntactic analysis and not on end-user applications.

The technology presented here has been developed for morphology-rich languages. It offers a framework for building language-specific analysers, and directly turn them into a wide range of useful programs. It shares

2 Infrastructure requirements

The infrastructure presented here meets the following requirements: it separates between language-independent and language-specific data and constructs; it works the same for all languages, thereby letting the linguists focus on the actual linguistics, and ease comparison and cooperation across languages; it uses standard and widely available tools as far as possible; it uses and depends on open-source LT tools, either exclusively or as an alternative; it is platform neutral (but requires un*x for compilation); it supports building a large number of scientific and end-user tools, both normative and descriptive; it supports dialect variation in a standardised way; it integrates with or exports end-user tools for different platforms (e.g. spellers, hyphenators, grammar checkers, for both open-source and closed-source host systems and applications); and finally: it supports reuse of code as much as possible.

[1]`http://www.ethnologue.com/statistics/status` - the sum of languages with an EGIDS value 0-5: 2216.
[2]`http://nltk.org/`
[3]`http://opennlp.apache.org/`
[4]`http://moin.delph-in.net/`
[5]`http://www2.parc.com/isl/groups/nltt/xle/`

This is a tall order. While everything is not yet implemented, most of the items on the list are already in place and working well.

3 Technical details

In this section we will present the strategy employed to handle a large number of languages in a consistent way, while at the same time allowing language-specific flexibility — and all of it hopefully without increasing the maintenance burden regarding the infrastructure.

Although many of the points mentioned below are standard good practice of software development, it bears repeating. And to our knowledge nothing like this has been done for morphology-rich minority languages before, targeting both the scientific community and the language communities to the extend and breadth done here. The importance of good software design and an infrastructure supporting code reuse is crucial for language communities with limited resources which can not afford to support several different language technology projects and competing technologies. The choices are made on the ground that it gives double return of the investments: both grammars and tools for linguistic analysis, and software products for the end user community.

3.1 Layout and design

The basic model for the build and configuration infrastructure consists of two parts:

- a central core, containing resources shared by all languages
- a templating system, by which new functionality and new resources are propagated to all languages

The template contains both build files (automake files) and language resource template files. The language resources are copied once, either when a new language is initialised, or the first time a new language resource is made available in the template. The build files, on the other hand, are merged from the template whenever requested. A schematic view of the infrastructure can be seen in figure 1.

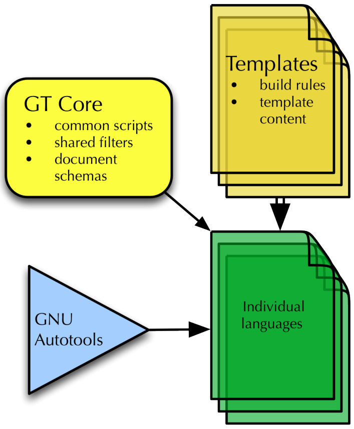

Figure 1: A schematic view of the new Giellatekno/Divvun infrastructure.

3.2 Software

The software used in the infrastructure can also be grouped in two. The first category contains the build tools — the tools used to configure, test, enable and create the build environment. The second category covers the language technology tools — the tools used to build analysers, proofing tools and more, using the linguistic resources and source code.

3.2.1 Build software

One goal with the infrastructure is to use a standard set of build tools combined with established practises for source code layout and organisation. This gives several benefits. It makes it easy for any other open-source project to include our resources, since the build system follows a known pattern. It makes it easier for newcomers, even those who have never seen a software project before - the source code is clearly organised in logical units, and the commands to build and test the tools are few, and the same for all languages.

We base the infrastructure build system on the GNU Autotools[6]: *automake* and *autoconf* and their supporting tools. The tool set is well established in the open-source community, has broad support and handles most cross-platform issues quite well.

3.2.2 Language-technology software

From the very beginning most of the work has been done on languages with rich morphology, and it has been important that the linguistic resources should be usable for as many purposes as possible. Another aspect has been that all the initial languages were minority languages with very little available corpus data. All this is to say that statistical methods and tools have *not* been seen as alternatives. Instead morphological analysis and disambiguation, as well as syntactic analysis, has been rule-based. The basic observation is that there is little corpus data, but as long as there are speakers there is also a grammar and a lexicon — and that is (almost) all that is needed for a rule-based approach.

Initially all work on morphological analysis were done using Xerox' finite state transducer tools, and still today they dominate our daily work. But the Xerox FST tools are closed source, fixing bugs depends on others, the licensing terms have changed and might change again, and who knows what may happen in the future.

During the last years there has been a continuous effort at the Helsinki university to develop a source-code compatible, open-source alternative to the Xerox tools, the HFST tools (Lindén et al., 2011)[7]. These tools are now quite stable and well suited as an alternative, and at the same time they provide more features than the Xerox tools, most notably weighted transducers.

The infrastructure uses the Xerox tools as the default tool set, but has a parallel build system made for hfst. Turning *off* Xerox and *on* HFST is just two options away at configuration time. It is even possible to have both tool sets enabled at the same time - this will cause the build system to produce double sets of transducers, one set for each technology. This gives the added benefit of double error checking of the source code — and it also makes it possible to compare the two tool sets side by side (more on this further down).

For morphological disambiguation and higher level analysis we use Constraint Grammar

[6]http://en.wikipedia.org/wiki/GNU_build_system
[7]http://hfst.sf.net/

((Karlsson, 1990) as implemented by the VislCG3 project[8]). VislCG3 is open source as well, and together with HFST this gives a complete open-source tool chain from source code to high-quality analysers including dependency analysis.

3.3 Support for best practices: in-source documentation and unit testing

One of the problems with large software projects (which the development of a complete morphological and syntactic description and a large-scale lexicon is) is that the system becomes so complex that no-one can overview it in full and with all details. This is true both for the linguistic source code and for the infrastructure.

The infrastructure supports a simple system for writing documenting comments as part of the source code. The system is inspired by numerous systems for other programming languages, many of which goes back to (Knuth, 1984). The generated documentation is converted to html, and viewable online.

In addition to in-source documentation, the infrastructure also supports specifying test data as part of the source code. This becomes essentially a unit-test-like setup (Huizinga and Kolawa, 2007): when you write a new inflectional lexicon for a new inflectional class, you can start by spelling out the wanted behaviour in the form of test cases. This also helps with refactoring the code.

The morphological testing is done using a tool developed as part of the Apertium[9] Quality Assurance project[10], but further developed to fit the needs of this infrastructure. The test tool supports bidirectional testing of the morphologies (both generation and analysis), and is thus well suited to test for overgeneration and errors in the morphology and morphophonology. The test tool is integrated with the generic testing facilities in Autotools.

3.4 Availability and sustainability

Going open-source is not only a generally good principle, it is a prerequisite for long-term sustainability and control over the language technology resources developed for minority and lesser resourced languages. To that end, all the required tools for the infrastructure are open source — or an open-source alternative is available via a configuration option — and all the linguistic resources themselves are of course open source[11]. Each language can set its own licence policy, but we encourage everybody involved to choose as open a license as possible.

As far as possible, we also try to support and develop end-user tools for open-source applications and solutions. While established tool sets that are taken for granted, like spellers, can be integrated system-wide on most platforms, newer or more advanced language technology solutions like speech recognition or grammar checkers are blocked[12].

The target audience for the infrastructure is first and foremost linguists and computation linguists. The tools are accessed and used from a Unix command line, and basic familiarity with

[8]`http://beta.visl.sdu.dk/cg3.html`

[9]`http://www.apertium.org`

[10]`http://wiki.apertium.eu/index.php?title=Session_7:_Data_consistency_and_quality&oldid=2173`

[11]`http://divvun.no/doc/infra/GettingStarted.html`

[12]On some new platforms it isn't even possible to install keyboards for your own language, if you want to follow the license of the platform.

Development status keyword	Explanation
started	some initial work has been done, but the lexical content is very limited
developed	the morphological description is well on its way, the lexicon is substantial
large	the morphological description is more or less done, the lexicon is big
production	good coverage, complete grammatical description, used for end user tools

Table 1: Brief explanation of the development status categorisation.

unix systems is assumed. Furthermore, as has probably become clear, the main focus is on text processing applications, although new applications and data types will certainly be added in the future.

4 Language coverage

Here we list the languages presently in our infrastructure, along with their development status. (briefly described in table 1).

Production: Finnish, Lule Sami, Northern Sámi, Kalaallisut, Southern Sami,
Large: Erzya, Faroese, Norwegian Bokmål
Developed: Cornish, Eastern Mari, Komi-Zyrian, Kven Finnish, Iñupiaq, Old Norse, Somali
Started: Amharic, Buriat, Guaraní, Inari Sami, Ingrian, Inuktitut, Karelian, Kildin Sami, Kinyarwanda, Komi-Permyak, Livonian, Livvi, Moksha, Nenets, Ojibwe, Pite Sami, Romanian, Skolt Sami, Tagalog, Tigrinya, Udmurt, Ume Sami, Veps, Võro, Western Mari

5 The tools produced by the infrastructure

The infrastructure will build a number of tools for all the covered languages (the quality of the tools varies a lot depending on the development stage of each language).

The most important compiled resources are **morphological analysers and generators**, both normative and descriptive ones. In addition to being at the core of the textual analysis pipeline, these are also used as parts of electronic dictionaries, language learning tools, online services such as paradigm and word-form generation, and as important blocks in machine translation.

The infrastructure will compile any CG3 rules file into **syntactic analysers**, possibly with **dependency analysis** if the source file exists (for many of the languages covered, it is quite possible to reuse the dependency analysis (Antonsen et al., 2010)).

A classic application (Oflazer, 1996)) of transducers is as **proofing tools** - spellers and hyphenators. At the time of writing, spellers based on a combination of HFST and Voikko[13] are produced by the infrastructure. For most of the languages listed in table 3 this is a first, and the value for the user community is very real and direct.

The proofing tool example also illustrates the double benefit of working rule-based: grammatical models are developed, which often means new insight into the language, or that knowledge that earlier was only indirectly described needs to be made explicit. That is, the grammatical description of the language often becomes better and more explicit, one gets a large lexicon usable as a starting point for many different projects and research efforts, and at the same time — literally — one can give back to the language community in the form of usable and valuable tools. This is a great inspiration for the work described in this paper.

[13]`http://voikko.sf.net/`

System: Language	HFST	Xerox
Greenlandic	1 h 11 min	37 min 12 s
Faroese	6 min 25 s	3 min 55 s
Lule Sámi	2 min 44 s	30.9 s
South Sámi	2 min 4 s	11.1 s
Finnish	18.7 s	16.4 s
Komi-Zyrian	59.3 s	4.6 s
Moksha	39.9 s	2.7 s
Eastern Mari	36.8 s	2.7 s
Western Mari	36.7 s	2.7 s
Udmurt	29.3 s	0.9 s
Erzya	26.7 s	2.1 s
Skolt Sámi	26.2 s	3.0 s
Voru	22.7 s	1.5 s
Ingrian	21,6 s	1,5 s
Livvi	21.1 s	1.6 s
Inari Sámi	18.8 s	1.3 s
Pite Sámi	16.0 s	3.6 s
Kildin Sámi	8.1 s	1.4 s
Kven	7,8 s	1,1 s
Livonian	4.6 s	0.8 s
Veps	4.3 s	0.9 s
Nenets	4.3 s	0.9 s
Khanty	3.7 s	0.9 s
Nganasan	3.4 s	0.8 s

Table 2: The compilation speed of the languages in the system (times given in seconds unless otherwise noted)

6 Some interesting figures for comparative computational linguistics

One of the interesting features we gain from a uniform, standardised suite for building and using computational linguistic descriptions is easy access to comparable evaluations of the implementations. We can measure the quality of the analysers by coverage, by the ambiguity potential of the languages, we can compare features like morphological complexity of the languages by measuring the morpheme to word ratios. We can measure the computational complexity of languages in terms of used processing power, the physical sizes of the models built and so forth. In the following we lay out and present a few of such measurements that can be attained by very simple testing leveraging the strength of the standard build suites we have created.

The computational complexity of the models can be measured using two very practical figures: build time of the language models and the physical size. In figure 2 we show the build times of the systems in our repository, both using commercial tools and open source ones. In that table, the horizontal lines separate the systems on basis of completeness: the first set of languages is considered stable and widely usable, and the second set work in progress. Other than that, the languages are in alphabetical order of ISO 639-3 language codes.

These need to be contrasted with the initial, linguistic sizes of the models from table 3. In this table we show how big the dictionaries are in terms of words in the dictionary and in terms of approximate[14] suffix morphemes. In table 3 we show the actual sizes of the language models as resulting computational objects, measured both in bytes and in components of the data structure holding the data.

One rough figure to measure readiness of a linguistic description is *coverage*, which tells how much of a given text would be recognised as something using the given language model, without taking correctness into consideration. Formally it is given as $\mathrm{Cov} = \frac{\mathrm{Analysed}}{\mathrm{tokens}}$. In table 3 we measure each language's coverage against up to the first 10,000,000 tokens in the native language Wikipedia, where one exists.

[14]The actual number is not apparent from the count of morpheme-like elements as the formalism may require to treat combinatorical branching as zero-morpheme

Units	Roots	Affixes	States	Edges	On-Disk	Corpus	Missing	Coverage
Language					Size			
Greenlandic			398,622	13,018,901	170 MiB	27674	7650	72.4 %
South Sámi	88,239	3,090	92,191	215,501	6 MiB			
Faroese	87,567	1,460	80,969	1,282,798	32 MiB			81.5 %
Lule Sámi	73,101	2,943	103,504	260,401	7 MiB			
Finnish	71,761	55,895	179,891	403,944	9.6 MiB			85.1 %
Erzya	106,550	2,860	65,571	174,427	5.8 MiB	63028	10379	83.5 %
Komi-Zyrian	88,734	1,336	56,146	147,618	4.4 MiB	98573	17286	82.5 %
Moksha	59,377	311	35,013	86,086	2.9 MiB	52803	28883	45.3 %
Eastern Mari	56,684	180	35,139	87,578	2.8 MiB	212393	59371	72.0 %
Western Mari	53,236	180	28,734	70,890	2.2 MiB	219337	105857	51.7 %
Udmurt	37,101	84	35,986	65,296	1.8 MiB	107453	63096	41.3 %
Livvi	10,936	173	9,476	19,476	533 KiB			
Voru	15,799	56	10,204	23,855	642 KiB			
Inari Sámi	3,560	280	7,930	12,595	349 KiB			
Ingrian	2,032	1,641	7,922	16,349	377 KiB			
Kildin Sámi	1,840	139	2,951	5,089	157 KiB			
Pite Sámi	1,470	318	3,184	6,281	174 KiB			
Kven	662	131						
Skolt Sámi	583	426	2,761	4,718	120 KiB			

Table 3: Three aspects of the automata: number of relevant morphs and affixes, graph properties, and corpus coverage measured on a Wikipedia corpus, where available

6.1 Platform for linguistically oriented comparisons

One of the potential use cases for a standardised repository of closely related languages would be for diachronic and comparative linguistics. It would not be difficult to formulate finite-state models to verify theories and similarities of related languages, e.g. in style of (Wettig et al., 2011).

Another immediate advance for a standardised structure like this is that we can build rule-based machine-translation systems easier. In shallow-transfer systems like Apertium (Forcada et al., 2011), when building closely related languages like in the Uralic family, we may get quite far by only writing transfer code between lemmas, with no other rules, or very minimal case mappings.

7 Conclusion

In this article we have presented our work with the Divvun and Giellatekno new infrastructure. It is in daily use for roughly half the languages being worked on, and for many of the production languages. Both our experience and the feedback from linguists and other users have shown that it is a definite improvement over the old one. It is based on open-source tools, and contain almost only open-source resources. We welcome many more languages, and we are confident we can handle them.

Acknowledgments

Thanks to our colleagues, especially Jack Rueter, for very valuable feedback and comments.

References

Antonsen, L., Trosterud, T., and Wiechetek, L. (2010). Reusing Grammatical Resources for New Languages. In Calzolari, N., Choukri, K., Maegaard, B., Mariani, J., Odijk, J., Piperidis, S., Rosner, M., and Tapias, D., editors, *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, Valletta, Malta. European Language Resources Association (ELRA).

Broda, B., Marcińczuk, M., and Piasecki, M. (2010). Building a Node of the Accessible Language Technology Infrastructure. In *Proceedings of the Seventh conference on International Language Resources and Evaluation (LREC'10)*.

Cunningham, H., Humphreys, K., Gaizauskas, R., and Wilks, Y. (1997). Software infrastructure for natural language processing. In *Proceedings of the fifth conference on Applied natural language processing*, ANLC '97, pages 237–244, Stroudsburg, PA, USA. Association for Computational Linguistics.

Cunningham, H., Maynard, D., Bontcheva, K., Tablan, V., Aswani, N., Roberts, I., Gorrell, G., Funk, A., Roberts, A., Damljanovic, D., Heitz, T., Greenwood, M. A., Saggion, H., Petrak, J., Li, Y., and Peters, W. (2011). *Text Processing with GATE (Version 6)*. Gate.

Federmann, C., Giannopoulou, I., Girardi, C., Hamon, O., Mavroeidis, D., Minutoli, S., and Schröder, M. (2012). META-SHARE v2: An Open Network of Repositories for Language Resources including Data and Tools. In Calzolari, N., Choukri, K., Declerck, T., Doğan, M. U., Maegaard, B., Mariani, J., Odijk, J., and Piperidis, S., editors, *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey. European Language Resources Association (ELRA).

Forcada, M. L., Ginestí-Rosell, M., Nordfalk, J., O'Regan, J., Ortiz-Rojas, S., Pérez-Ortiz, J. A., Sánchez-Martínez, F., Ramírez-Sánchez, G., and Tyers, F. M. (2011). Apertium: a free/open-source platform for rule-based machine translation. *Machine Translation*.

Huizinga, D. and Kolawa, A. (2007). *Automated Defect Prevention: Best Practices in Software Management*. Wiley.

Karlsson, F. (1990). Constraint Grammar As A Framework For Parsing Running Text. *Proceedings of the 13th International Conference on Computational Linguistics*, pages 168–173.

Knuth, D. E. (1984). Literate Programming. *The Computer Journal*, 27(2):97–111.

Lindén, K., Axelson, E., Hardwick, S., Pirinen, T., and Silfverberg, M. (2011). Hfst—framework for compiling and applying morphologies. *Systems and Frameworks for Computational Morphology*, pages 67–85.

Loper, E. and Bird, S. (2002). NLTK: the Natural Language Toolkit. In *Proceedings of the ACL-02 Workshop on Effective tools and methodologies for teaching natural language processing and computational linguistics - Volume 1*, ETMTNLP '02, pages 63–70, Stroudsburg, PA, USA. Association for Computational Linguistics.

Oflazer, K. (1996). Error-tolerant finite state recognition with applications to morphological analysis and spelling correction. *COMPUTATIONAL LINGUISTICS*, 22:73–89.

Trosterud, T. (2012). A restricted freedom of choice: Linguistic diversity in the digital landscape. *Nordlyd*, 39(2):89–104.

Váradi, T., Krauwer, S., Wittenburg, P., Wynne, M., and Koskenniemi, K. (2008). CLARIN: Common Language Resources and Technology Infrastructure. In Calzolari, N., Choukri, K., Maegaard, B., Mariani, J., Odijk, J., Piperidis, S., and Tapias, D., editors, *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)*, Marrakech, Morocco. European Language Resources Association (ELRA). http://www.lrec-conf.org/proceedings/lrec2008/.

Wettig, H., Hiltunen, S., and Yangarber, R. (2011). MDL-based Models for Alignment of Etymological Data. In *Proceedings of RANLP: the 8th Conference on Recent Advances in Natural Language Processing, Hissar, Bulgaria*.