Automatic Testing and Evaluation of Multilingual Language Technology Resources and Components

Ulrich Schäfer, Daniel Beck

German Research Center for Artificial Intelligence (DFKI), Language Technology Lab
Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany
email: {ulrich.schaefer,daniel.beck}@dfki.de

Abstract

We describe SProUTomat, a tool for daily building, testing and evaluating a complex general-purpose multilingual natural language text processor including its linguistic resources (lingware). Software and lingware are developed, maintained and extended in a distributed manner by multiple authors and projects, i.e., the source code stored in a version control system is modified frequently. The modular design of different, dedicated lingware modules like tokenizers, morphology, gazetteers, type hierarchy, rule formalism on the one hand increases flexibility and re-usability, but on the other hand may lead to fragility with respect to changes. Therefore, frequent testing as known from software engineering is necessary also for lingware to warrant a high level of quality and overall stability of the system. We describe the build, testing and evaluation methods for LT software and lingware we have developed on the basis of the open source, platform-independent Apache Ant tool and the configurable evaluation tool JTaCo.

1. Introduction

The development of multilingual resources for language technology (LT) components is a tedious and error-prone task. Resources like tokenisers, morphologies, lexica, grammars, gazetteers etc. for multiple languages can only be developed in a distributed manner, i.e., many people work on different resources.

However, the resulting systems are supposed to deliver the same good recognition quality for each language. Dependencies of resources and subsystems may lead to suboptimal functioning, e.g. reduced recognition rates, of the overall systems in case of errors creeping in during the development process.

Hence, like in software engineering, testing and evaluation of the developed lingware resources has to be performed on a regular basis, both for quality assurance (QA) and comparability of results in different languages.

In this paper, we describe SProUTomat, a tool for daily building, testing and evaluating the complex general-purpose multilingual natural language text processor SProUT, its software components and lingware resources.

Independently of the SProUT system, many of the concepts and mechanisms described in the paper could be applied to any other resource-intensive natural language processing system.

After a brief introduction to SProUT and Apache Ant, we describe the four main components of SProUTomat: the build and compilation part, testing, evaluation and report generation. We conclude with a short summary.

2. SProUT

SProUT (Drożdżyński et al., 2004) is a general, multilingual multi-purpose natural language processor. SProUT comes with a powerful, declarative grammar formalism (XTDL) that combines finite-state techniques and typed feature structures—with structure sharing and a fully-fledged, efficiently encoded type hierarchy, in contrast to systems like GATE (Cunningham et al., 2002) that support only simple attribute-value pairs.

SProUT rules consist of regular expressions over typed feature structures[1]. A rule is matched against a sequence of input feature structures, e.g. filled by basic components like tokenizers, morphology or gazetteer lookup running on input text or, in more complex cases, XML input or even output from previous SProUT grammar stages.

The matching condition is unifiability of the input sequence with the expanded regular expression of the left hand side of a rule. In case of a match, unification is used to transport information from the matching left hand side to the (single) output feature structure on the right hand side of the rule. The resulting feature structures generated by analysing a text can e.g. be serialised to an XML document.

The SProUT system provides basic components such as tokenizers, morphologies, domain-specific gazetteers and a common TDL type hierarchy for languages such as English, German, French, Spanish, Greek, Japanese, Italian, Chinese, Polish and Czech. Moreover, named entity recognition and information extraction grammars in XTDL exist for most of these languages.

Each of the resources is maintained in source code checked in a version control system, and compiled into a specialized, binary representation for efficient processing at runtime.

SProUT has been used in many projects, e.g. for automatic hyperlinking, opinion mining, question answering and text mining for air traffic forecasts.

The main applications of SProUT are multilingual information extraction and named entity recognition in closed domains. However, the formalism can also be used to perform different duties, e.g. rule-based transformation of typed feature structures or XML objects, as described in (Frank et al., 2004).

3. Motivation

The need for an automatic build, testing and evaluation system became obvious when multiple projects became 'cus-

[1] The acronym SProUT stands for Shallow Processing with Unification and Typed feature structures.

tomers' of the SProUT system. Both software (at least in the early development phase) and lingware changed very frequently (check-ins several times per day in hot phases), which lead to frequent problems when compiling and running SProUT. Since that time and up today, multiple authors are developing multiple components and lingware in multiple projects on multiple domains in multiple languages.

4. Apache Ant

Apache Ant (http://ant.apache.org) is a standard open-source tool for automatic building and packaging complex software systems. On the basis of target descriptions in an XML configuration file, ant automatically resolves a target dependency graph and executes only the necessary targets.

The concept is similar to that of the Unix make tool. However, Ant inherits from Java (in which is implemented) the advantage that it is platform-independent. Moreover, many of the often-needed auxiliary tools e.g. zip, tar, gzip, bzip, jar, ftp, SQL, SMTP, XSLT, scp and file/dir operations, are built-in in Ant and work completely platform-independent. From XML, Ant inherits Unicode and a highly structured syntax, for which already nice tools exist, including Eclipse support and visualisation of target dependency graphs (Figure 3).

Powerful mechanisms like patternset, fileset, filemapper, filterset and filterchain allow to define re-usable sets of patterns e.g. for bunches of source files.

Ant functionality can be extended by (1) defining targets in the configuration file, (2) extending the ant syntax itself by pluggable Java classes that can easily be written, (3) calling external programs (processes) or BSF scripts.

Following is a small sample Ant build file containing a single target illustrating the structure of a project definition. The `depends` attribute in the target definition for `runtimejar` indicates that first the `compile` target (not shown here) has to be successfully executed once before the body of the target definition is executed.

```
<?xml version="1.0"?>
<project default="ide" name="SProUT">
 <description>
  This is the ant build file for SProUT.
 </description>

 <!-- load user-specific settings -->
 <property file="user.properties"/>
 ...

 <target name="runtimejar" depends="compile"
  description="Build runtime jar.">
  <jar jarfile="sprout-runtime.jar">
    <fileset dir="${classes.dir}">
     <patternset refid="jar.fileset"/>
    </fileset>
  </jar>
  <echo>done.</echo>
 </target>

</project>
```

5. SProUTomat

SProUTomat is an automatic build, testing and evaluation tool for linguistic resources and components that has been implemented for the SProUT system and its numerous multilingual resources. SProUTomat is used for daily building and testing the development and runtime system from the Java and lingware sources mainly for named entity and information extraction grammars from a version control system.

SProUTomat is an extension of the build mechanism for language technology components and resources we have developed for the SProUT system using Apache ant

5.1. Build and Compilation

Before testing and evaluating, a system has to be built, i.e. compiled from the sources checked out from the source control system. The Java program code compilation of SProUT is a straightforward task best supported by Ant. The case is, however, different for lingware sources (type hierarchy[2], tokeniser, morphology, gazetteer, XTDL grammars).

While the appropriate Java code compilation tasks know what a compiled class file is and when it has to be re-compiled (source code changes, dependencies), this has to be defined explicitly for lingware resources which Ant natively is not aware of. The `uptodate` task can be used to compare source files (.tdl in the following example) against their compiled version (.grm).

```
<uptodate property="tdl_input_is_uptodate"
          srcfile="${typehierarchy}.tdl"
       targetfile="${typehierarchy}.grm"/>
```

For each of the different lingware types, these source file dependencies are defined as are the calls to the dedicated SProUT compilers and parameters for their compilation.

Lingware-specific targets have common parameters and properties like `"lang"`, `"project"` or the lingware type that are used to locate e.g. the source and compiled files in the hierarchically defined directory trees or `"charset"` to specify encodings for source files to read.

Dependencies between different lingware types are handled by calls to defined sub-targets. Figure 1 shows the definition of the `compile_ne` target that calls four other compilation sub-targets. Each subtarget compiles only when necessary, and the `compile_ne` target itself depends on the `jar` target that provides working and up-to-date SProUT lingware compilers.

Besides the program and lingware compilation, many other targets exist e.g. to generate documentation, package runtime systems, start the integrated development environment etc.

Thus, using a single command, it is possible to compile the whole system including code and all dependent available linguistic resources, or to update it after changes in the sources.

[2] The SProUT formalism uses a subset of TDL (Krieger and Schäfer, 1994) that is compiled using the *flop* compiler of the PET system (Callmeier, 2000).

```
<!--Compiles all named entity grammar resources for a given language.-->
<!--usage : ./ant compile_ne -Dlang=en -Dsubgrammar=ne -Dproject=xyz -->
<target name="compile_ne" depends="jar"
  description="Compile all NE grammar resources for a given language.">
  <!--Description: Compiles all NE grammar resources for a given language. -->
  <!--Parameter: ${lang} : language code (ISO 639) of the ne grammar to compile.
  <!--Parameter: ${charset} : encoding of the extended gazetteer input file -->
  <!--Parameter: ${project} : name of the project. Default is "" -->
  <!--Parameter: ${subgrammar} : name of the subgrammar to compile.-->

  <!--default properties-->
  <property name="lang" value="en"/>  <!-- ISO 639 language code -->
  <property name="project" value=""/> <!-- default named entity grammar -->
  <property name="charset" value="ISO-8859-1"/> <!-- encoding -->

  <antcall target="compile_tdl"/>     <!-- common type hierarchy -->
  <antcall target="compile_tokenclass"/> <!-- tokeniser -->
  <antcall target="compile_extended_gazetteer"/>
  <antcall target="compile_grammar"/> <!-- i.e., XTDL grammar -->
</target>
```

Figure 1: A sample target definition: general named entity grammar compilation.

A dependency graph of the defined targets for SProUTomat is depicted in Figure 3.

An application of the built-in XSLT funtionality of Ant is e.g. OntoNERdIE (Schäfer, 2006) that has been integrated by simply applying three XSLT stylesheets for mapping instance and concept data from OWL ontologies to SProUT named entity recognition and information extraction resources.

5.2. Testing

The daily automatic testing and evaluation mechanism is an extension of the build procedure. SProUTomat first updates all program sources and linguistic resources from the version control system, and compiles them from scratch. For each language resource to test, a reference text is then analysed by the SProUT runtime system called through appropriate Ant targets. This checks for consistency of the sources. The test is only considered successful ('OK') if program code as well as all lingware sources compile successful and the text analysis runs successful.

5.3. Evaluation: JTaCo

The next step is comparison of the generated named entity and information extraction annotation against a gold standard. SProUTomat uses JTaCo (Bering et al., 2003) for the automatic evaluation and computation of precision, recall and f-measure. Details on the SProUTomat integration are discussed in (Bering and Schäfer, 2006). For the evaluation of English named entity grammars, the annotated corpus is e.g. taken from the MUC evaluation data (Grishman and Sundheim, 1996). For other languages for which no MUC annotations exist (e.g. German), a manually developed corpus is employed.

JTaCo strips off the annotation from the marked up corpus, sends it to the SProUT runtime processor, and compares the returned markup with the annotated corpus using a mapping between the two formats.

As an illustration, consider SProUT's NER markup and the MUC-6 annotation format for named entities. While MUC-6 foresees the following markup for time expressions,

```
<TIMEX TYPE="DATE">07-21-96</TIMEX>
```

SProUT would deliver structured output in a typed feature structure[3].

$$\begin{bmatrix} \textit{point} & \\ \text{SPEC} & \textit{temp-point} \\ \text{MUC-TYPE} & \textit{date} \\ \text{CSTART} & \texttt{"27"} \\ \text{CEND} & \texttt{"34"} \\ \text{SURFACE} & \texttt{"07-21-96"} \\ \text{YEAR} & \texttt{"1996"} \\ \text{MONTH} & \texttt{"07"} \\ \text{DOFM} & \texttt{"21"} \end{bmatrix}$$

JTaCo provides a graphical user interface that supports the user in defining mappings between annotation formats to compare so that it can be easily customised for comparison with other XML annotation formats. For the daily evaluation, a batch version of JTaCo based on predefined annotation mappings is used. An example of the computed precision and recall values over time can is included in Figure 4.

5.4. Report generation

Finally, a report (Figure 4) is generated and emailed to the developers with an overall status (OK or ERROR) for quick information on the testing result. The report also contains diagrams consisting of precision, recall and f-measure

[3] Transformation of typed feature structures and general XML markup is discussed in the context of the upcoming ISO standard in (Lee et al., 2004). Actually, SProUT's default XML output format is very close to the proposed ISO format for typed feature structures.

curves since beginning of regular measurements per language that visually give an overview of the resource development progress over time. To this end, the evaluation numbers are also added to a global evaluation database.

Further information sources like Ant target and Javadoc documentation as well as a visual dependency graph representation of the ant targets are generated automatically.

Although daily testing has been described above, the testing and report generation could be started at any time. A complete build from scratch, testing of four languages including Javadoc and generation of the runtime system plug-in into the Heart of Gold platform for deep-shallow integration (Callmeier et al., 2004) etc. takes less than 14 minutes, while only a few seconds are required after modification of a single resource.

The daily report was successful ('OK') in 93,5% of the working days during the last eight months, and, as can be seen from Figure 4, precision and recall could be improved.

6. Summary

We have presented a comprehensive tool for automatically testing and evaluating linguistic resources and language technology components. The system is in daily use since March 2005 and successfully helps to maintain the quality and reliability of the multilingual language processor with its various resources that are developed by many authors and used in several projects. The tool greatly helps to improve and accelerate the development - evaluation/comparison - refinement - cycle (Figure 2) and gives motivating feedback, such as raising recall and precision curves over time.

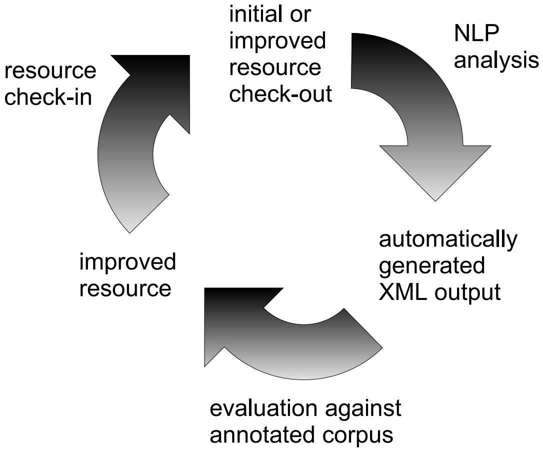

initial or improved resource check-out
NLP analysis
automatically generated XML output
evaluation against annotated corpus
improved resource
resource check-in

Figure 2: Quality assurance and development cycle for multilingual linguistic resources.

7. Acknowledgements

We would like to thank Christian Bering for developing the JTaCo tool, the SProUT grammar developers for their feedback, Witold Drożdżyński for extending the SProUT API to our needs and the LREC reviewers for helpful comments.

This work has been supported by a grant from the German Federal Ministry of Education and Research (FKZ 01IWC02).

8. References

Christian Bering and Ulrich Schäfer. 2006. JTaCo & SProUTomat – automatic evaluation and testing of multilingual language technology resources and components. Submitted paper.

Christian Bering, Witold Drożdżyński, Gregor Erbach, Clara Guasch, Petr Homola, Sabine Lehmann, Hong Li, Hans-Ulrich Krieger, Jakub Piskorski, Ulrich Schäfer, Atsuko Shimada, Melanie Siegel, Feiyu Xu, and Dorothee Ziegler-Eisele. 2003. Corpora and evaluation tools for multilingual named entity grammar development. In *Proceedings of Multilingual Corpora Workshop at Corpus Linguistics*, pages 42–52, Lancaster.

Ulrich Callmeier, Andreas Eisele, Ulrich Schäfer, and Melanie Siegel. 2004. The DeepThought core architecture framework. In *Proceedings of LREC-2004*, pages 1205–1208, Lisbon, Portugal.

Ulrich Callmeier. 2000. PET – A platform for experimentation with efficient HPSG processing techniques. *Natural Language Engineering*, 6(1):99–108.

Hamish Cunningham, Diana Maynard, Kalina Bontcheva, and Valentin Tablan. 2002. 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*.

Witold Drożdżyński, Hans-Ulrich Krieger, Jakub Piskorski, Ulrich Schäfer, and Feiyu Xu. 2004. Shallow processing with unification and typed feature structures – foundations and applications. *Künstliche Intelligenz*, 2004(1):17–23.

Anette Frank, Kathrin Spreyer, Witold Drożdżyński, Hans-Ulrich Krieger, and Ulrich Schäfer. 2004. Constraint-based RMRS construction from shallow grammars. In Stefan Müller, editor, *Proceedings of the HPSG-2004 Conference, Center for Computational Linguistics, Katholieke Universiteit Leuven*, pages 393–413. CSLI Publications, Stanford.

Ralph Grishman and Beth Sundheim. 1996. Message understanding conference - 6: A brief history. In *Proceedings of COLING-96*, pages 466–471.

Hans-Ulrich Krieger and Ulrich Schäfer. 1994. TDL – a type description language for constraint-based grammars. In *Proceedings of COLING-94*, pages 893–899.

Kiyong Lee, Lou Burnard, Laurent Romary, Eric de la Clergerie, Ulrich Schäfer, Thierry Declerck, Syd Bauman, Harry Bunt, Lionel Clément, Tomaz Erjavec, Azim Roussanaly, and Claude Roux. 2004. Towards an international standard on feature structure representation (2). In *Proceedings of the LREC-2004 workshop on A Registry of Linguistic Data Categories within an Integrated Language Resources Repository Area*, pages 63–70, Lisbon, Portugal.

Ulrich Schäfer. 2006. OntoNERdIE—mapping and linking ontologies to named entity recognition and information extraction resources. In *Proceedings of the 5th International Conference on Language Resources and Evaluation LREC-2006*, Genoa, Italy, 5.

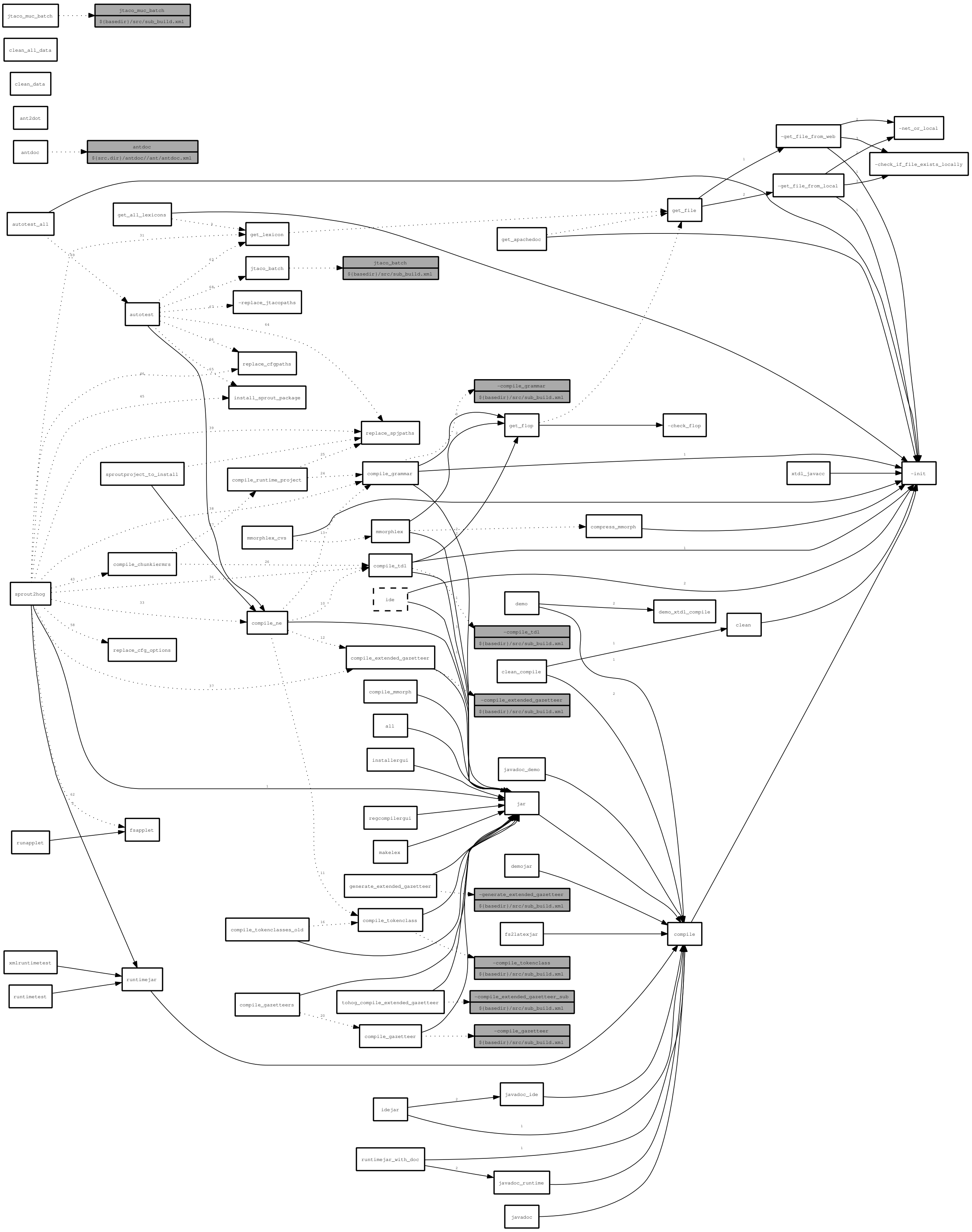
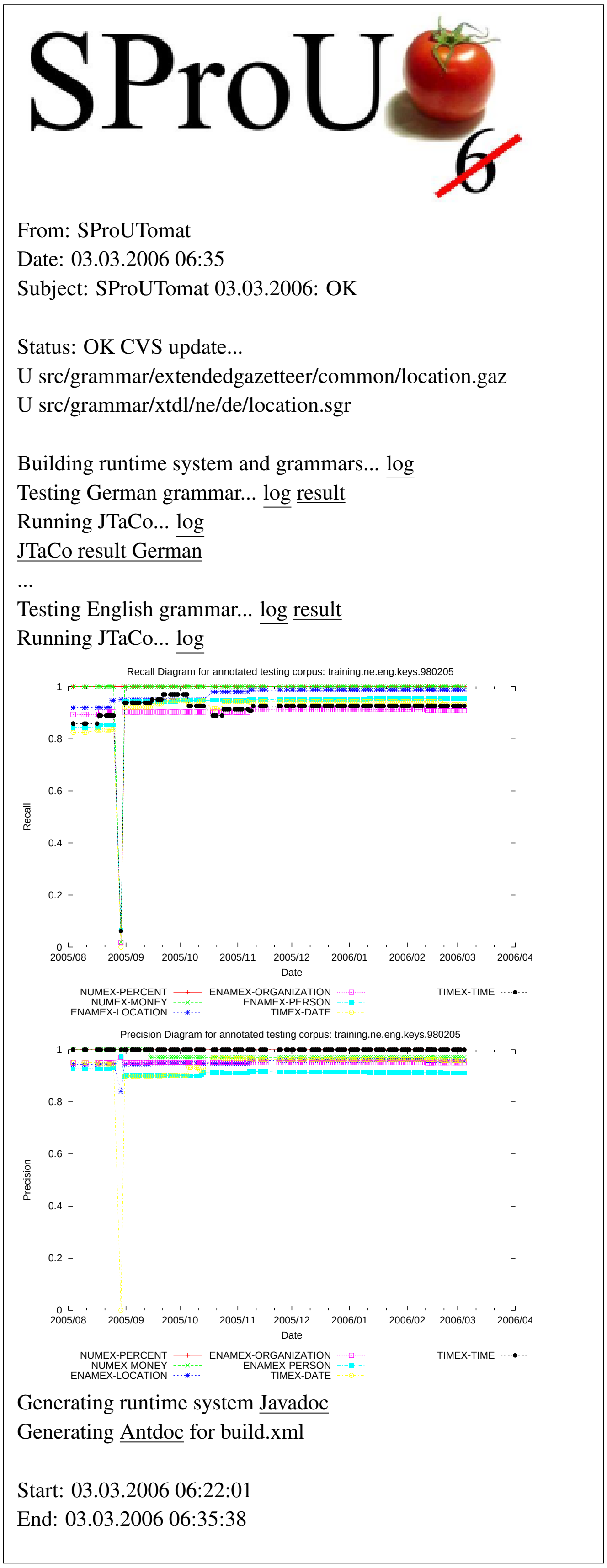


Figure 3: Graph of build, test and evaluation target dependencies generated with ant2dot.xsl (http://ant2dot.sourceforge.net). Solid lines indicate direct target dependencies, dotted lines indicate contained sub-target calls.



From: SProUTomat
Date: 03.03.2006 06:35
Subject: SProUTomat 03.03.2006: OK

Status: OK CVS update...
U src/grammar/extendedgazetteer/common/location.gaz
U src/grammar/xtdl/ne/de/location.sgr

Building runtime system and grammars... log
Testing German grammar... log result
Running JTaCo... log
JTaCo result German
...
Testing English grammar... log result
Running JTaCo... log

Generating runtime system Javadoc
Generating Antdoc for build.xml

Start: 03.03.2006 06:22:01
End: 03.03.2006 06:35:38

Figure 4: A report generated by SProUTomat (slashed).