A Multilingual Content Production Tool for the Semantic Web

Witold Drożdżyński, Hans-Ulrich Krieger, Jakub Piskorski, Ulrich Schäfer
DFKI GmbH
Stuhlsatzenhausweg 3,
66123 Saarbrücken, Germany

{witold,krieger,piskorsk,uschaefer}@dfki.de

1. INTRODUCTION

Automatic content extraction from unrestricted textual data constitutes a core technology for semantic web services. Intelligent content extraction must furthermore address the pecularities of the medium, i.e., must analyze natural language to a certain depth, in order to go beyond the realm of pure keyword-base approaches. This demo presents SProUT – a novel general-purpose multilingual information extraction (IE) platform [1]. The main motivation for its development comes from the need of having one modular system for multilingual & domain-adaptive IE, which is portable across different platforms. We also had to find a balance between efficiency and expressiveness of the grammar formalism.

SProUT is equipped with a set of reusable online processing components for basic linguistic operations, ranging from tokenization to text coreference resolution. They can be combined into a pipeline that produces several streams of linguistically annotated structures, which on the other hand serve as an input for the grammar interpreter, applied at the next stage.

Currently SProUT has been adapted to processing 11 languages, including major Germanic, Romance, Slavonic, and Asian languages. It has been deployed as the core IE component in several industrial and research projects [1]. In our presentation we showcase the development of the NE-recognition engine on top of SProUT for German and Polish [2].

2. GRAMMAR FORMALISM

The grammar formalism in SProUT is a blend of very efficient finite-state techniques and unification-based formalisms, guaranteeing expressiveness and transparency. To be more precise, a grammar in SProUT consists of pattern/action rules, where the LHS of a rule is a regular expression over typed feature structures (TFS) with functional operators and coreferences, representing the recognition pattern, and the RHS of a rule is a TFS specification of the output structure. Coreferences express structural identity, create dynamic value assignments, and serve as a means of information transport. Functional operators provide a gateway to the outside world, and they are utilized for introducing complex constraints in the rules, for forming the output of a rule, and for integrating external processing components.

Grammars consisting of such rules are compiled into extended finite-state networks with rich label descriptions (TFSs). For their efficient processing, a handful of methods going beyond standard finite-state techniques have been introduced [3]. Grammar rules can even be recursively embedded, which in fact provides grammarians with a context-free formalism. The following rule for the recognition of prepositional phrases gives an idea of the syntax of the grammar formalism:

```
pp :>  morph &  [  POS Prep, SURFACE #prep,
                   INFL [CASE #c ] ]
       (morph & [  POS Adjective,
                   INFL [  CASE #c,
                           NUMBER #n,
                           GENDER #g ] ] ) *
       (morph & [  POS Noun, SURFACE #noun1,
                   INFL [  CASE #c,
                           NUMBER #n,
                           GENDER #g ] ] )
       (morph & [  POS Noun, SURFACE #noun2,
                   INFL [  CASE #c,
                           NUMBER #n,
                           GENDER #g ] ] ) ?
-> phrase & [  CAT pp, PREP #prep,
               AGR agr & [   CASE #c,
                             NUMBER #n,
                             GENDER #g]
               CORE_NP #core_np]],
where #core_np=Append(#noun1," ",#noun2).
```

The first TFS matches a preposition. It is followed by zero or more adjectives. Finally, one or two noun items are consumed. The variables `#c`, `#n`, `#g` establish coreferences, expressing the agreement in case, number, and gender for all matched items (except for the initial preposition item which solely agrees in case with the other items). The RHS of the rule triggers the creation of a TFS of type phrase, where the surface form of the matched preposition is transported into the corresponding slot via the variable `#prep`. The value for the attribute `CORE_NP` is created through a concatenation of the matched nouns (variables `#noun1` and `#noun2`). This is realized via a call to functional operator `Append`.

3. STRUCTURED I/O INTERFACES

A generic XML input and output interface supports integration of foreign NLP components as input source for SProUT grammars, as well as XML output of SProUT results, including optional XSL transformation. SProUT grammars can also be cascaded using this XML interface or using the native feature structure interface (see section 4).

There are several ways in which SProUT can use and produce standard Semantic Web formats. (1) Via the described XML interface, RDF or other XML markup can be used as input for grammars. (2) RDF and XML markup can be generated by SProUT, using the XSLT output interface. (3) At

European Conference on Knowledge Engineering and Knowledge Management (EKAW'04), October 5–8, 2004, Northamptonshire, UK.

compile time, RDF definitions, e.g., for ontologies or taxonomies, can be compiled into TDL type definitions (TDL is the underlying typed feature formalism) [4] which can then be referred to in grammars or gazetteer definitions, turning SProUT into a declarative and hence elegant ontology-text interface device.

4. SDL

SProUT is shipped together with SDL, a description language that supports the declarative specification of NLP systems [5]. The building blocks of an SDL expression are existing modules, such as tokenizer, gazetteer, or named entity grammars. These modules can be connected via three operators, expressing sequence, parallelism, and unrestricted repetition. The SDL compiler generates Java codes which might be executed directly or integrated into a larger system. Interaction between modules is decoupled by a mediator. The usual control flow in a shallow cascaded system can be realized using the sequence operator. Other more advanced architectures utilize the other two operators [6].

5. DEVELOPMENT ENVIRONMENT

SProUT comes with an integrated graphical development and testing environment. The grammars can be either created in a TDL or in an XML editing mode, and can be visualized in a graphic mode. The grammar GUI resembles state-of-the-art development environments for programming languages, e.g., errors and warnings listed in the error message window are linked to the corresponding piece of grammar in the editor. Several user interfaces for inspecting the output of the linguistic processing components and for testing the grammars are provided, as shown in figure 1.

6. ACKNOWLEDGEMENTS

The development of the presented demo has been partially funded by the German BMBF, under grant no. 01 IW C02 (project QUETAL), and by additional non-financed personal effort of the authors.

7. REFERENCES

[1] W. Drożdżyński, H.-U. Krieger, J. Piskorski, U. Schäfer, F. Xu. *Shallow Processing with Unification and Typed Feature Structures – Foundations and Applications*. In German AI Journal Künstliche Intelligenz, Vol. 1/04, www.kuenstliche-intelligenz.de/archiv/2004_1/sprout-web.pdf, 2004.

[2] J. Piskorski. *Rule-based Named-Entity Recognition for Polish*. In Proceeding of the Workshop on Named-Entity Recognition for NLP Applications held in conjunction with IJNLP 2004, Sanya, Hainan Island, China.

[3] H.-U. Krieger, W. Drożdżyński, J. Piskorski, U. Schäfer, F. Xu. *A Bag of Useful Techniques for Unification-Based Finite-State Transducers*. In Proceedings of KONVENS 2004, Vienna, Austria.

[4] Hans-Ulrich Krieger and Ulrich Schäfer. *TDL - A Type Description Language for Constraint-Based Grammars*. In Proceedings of the 15th COLING, 893-899, 1994.

[5] Hans-Ulrich Krieger. *SDL - A Description Language for Building NLP Systems*. In Proceedings of the HLT-NAACL Workshop on the Software Engineering and Architecture of Language Technology Systems (SEALTS), 84-91, 2003.

[6] Anette Frank. *Constraint-based RMRS Construction from Shallow Grammars*. In Proceedings COLING, 2004.

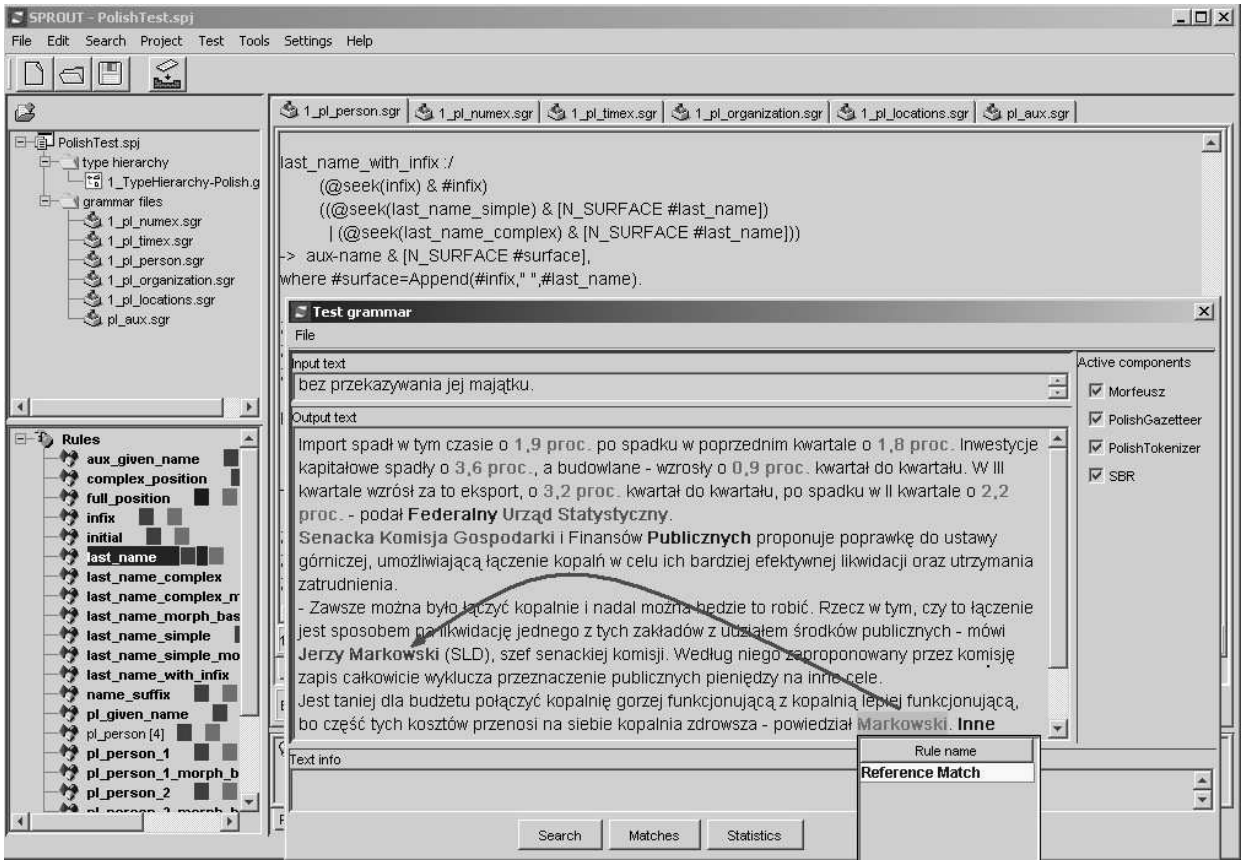

Figure 1. The SProUT IDE