Shallow Processing with Unification and Typed Feature Structures – Foundations and Applications

Witold Drożdżyński, Hans-Ulrich Krieger,
Jakub Piskorski, Ulrich Schäfer, Feiyu Xu

We present *SProUT*, a platform for the development of multilingual shallow text processing systems. A grammar in *SProUT* consists of a set of rules, where the left-hand side is a regular expression over typed feature structures (TFSs), representing the recognition pattern, and the right-hand side a TFS, specifying how the output structure looks like. The reusable core components of *SProUT* are a finite-state machine toolkit, a regular compiler, a finite-state machine interpreter, a typed feature structure package, and a set of linguistic processing resources. Several applications which make use of *SProUT* are presented. The system is implemented in Java and C(++), and runs under both MS Windows and Linux.

1 Introduction

Nowadays, we are witnessing an ever-growing trend of deploying lightweight linguistic analysis for solving problems that deal with the conversion of the vast bulk of raw textual information from myriads of digital data repositories into structured and valuable knowledge. Recent advances in the areas of information extraction, text mining, and textual question answering demonstrate the benefit of applying shallow text processing (STP) techniques, which are assumed to be considerably less time-consuming and more robust than deep processing systems, but are still sufficient to cover a broad range of linguistic phenomena.

This article also gives a walkthrough on the foundations and applications of *SProUT* (Shallow Processing with Unification and Typed feature structures), a novel platform for the development of multilingual STP systems. It consists of several linguistic processing resources which can be coupled in a flexible way for building higher-level linguistic engines, and provides an integrated grammar development and testing environment.

The motivation for developing *SProUT* comes from the need to have a system that (i) allows a flexible integration of different processing modules and (ii) to find a good trade-off between processing efficiency and expressiveness of the formalism. On the one hand, very efficient *finite-state* (FS) devices have been successfully applied to real-world applications. On the other hand, *unification-based grammars* (UBGs) are designed to capture fine-grained syntactic and semantic constraints, resulting in better descriptions of natural language phenomena. In contrast to FS devices, unification-based grammars are also assumed to be more transparent and more easily modifiable. The idea of *SProUT* is to take the best of these two worlds, having a FS machine that operates on typed feature structures (TFSs). I.e., transduction rules in *SProUT* do not rely on simple atomic symbols, but instead on TFSs, where the left-hand side (LHS) of a rule is a regular expression over TFSs, representing the recognition pattern, and the right-hand side (RHS) is a TFS, specifying the output structure. Consequently, equality of atomic symbols is replaced by *unifiability* of TFSs and the output is constructed using TFS *unification* w.r.t. a type hierarchy.

The article is structured as follows. Section 2 presents related work, viz., extended FS devices and unification-based grammars. After that, section 3 describes the formalism, starting with the building blocks (TFS, type definition, type hierarchy) and ending in regular expressions over TFSs. The architectural framework and the core components are discussed in section 4. Finally, section 5 focuses on various applications of *SProUT* in both research and industrial context.

2 Related Work

Both finite-state devices and unification-based grammars have influenced the shallow TFS formalism which will be presented in section 3.

2.1 Finite-State Devices

The pure finite-state-based STP approaches have proved to be very efficient in terms of processing speed. [23] present SPPC, a highly efficient system, which uses cascades of simple finite-state grammars, based on a small number of basic predicates. Complex constraints can not be encoded in the FS device. The idea of using more complex annotations on the transitions of FS automata has been considered in SMES [18] which uses regular grammars with predicates over morphologically analyzed tokens. These predicates inspect arbitrary properties of the input tokens, like part of speech or inflectional information. [30] introduce arbitrary predicates over symbols and discuss various operations on finite-state acceptors and transducers. They observe that automata with predicates generally have fewer states and transitions. However, the discussed automata only operate on symbols of a finite input alphabet. As a drawback of using too many or too complex predicates, standard optimization techniques are hardly applicable.

In the last few years, several cascaded FS-based systems have been developed for information extraction tasks. The most successful systems provide high-level specification languages for grammar writing. The pioneering FASTUS system [10] uses CPSL (Common Pattern Specification Language). The more recent GATE system [6] provides JAPE (Java Annotation Patterns Engine), which is similar in spirit to CPSL and admittedly borrows its main features from CPSL. A CPSL/JAPE grammar contains pattern-action rules. The LHS of a rule is a regular expression over atomic feature-value constraints (the recognition part), while the RHS is a so-called *annotation manipulation statement* for output production, which calls native code (e.g., C or Java), making rule writing difficult for non-programmers. Furthermore, even though there is a mechanism for variable binding which is responsible for copying values into the RHS, this mechanism is not capable of declaratively describing coreferences among the rule elements.

2.2 Unification-Based Grammars

Since the late seventies, UBG formalisms have become an important paradigm in natural language processing and computational linguistics. In the beginning, unification was employed as the primary constraint solving mechanism, hence the term *unification-based grammars*. Nowadays, this family of formalisms is often characterized through the more general notion *constraint-based*.

Their success stems from the fact that they can be seen as a *monotonic*, high-level representation language on which a parser/generator or a uniform type deduction mechanism acts as the inference engine. One of the main advantages of such formalisms is that they provide a *declarative* (as opposed to procedural) representation of linguistic knowledge, i.e., one must only specify the knowledge which participates in the constraint solving process, instead of anticipating the order in which the constraints are applied.

The representation of as much linguistic knowledge as possible through a unique data type called *feature structure* allows the integration of different description levels, spanning phonology, syntax, and semantics. Here, the feature structure itself serves as the abstract interface between the different strata which can thus be accessed and constrained at the same time. Central to feature structures is an operation which *combines* the information from two feature structures into a single structure, but also determines the *satisfiability* of the resulting description: *unification*.

Informally, a feature structure can be seen as a collection of feature-value pairs, where a feature expresses a functional property and the value of a feature might again be a feature structure (or an atom), thus allowing for recursive embeddings. An important characteristic of feature structures is that they allow for *coreference* constraints, meaning that two features share exactly one common value. This concept allows for the transport of information and is exhaustively used in *SProUT* grammar rules, where features on the LHS share values with other features on the RHS.

Feature structures can also be given a *type* which ultimately leads to a *typed feature structure*. First of all, a type can be seen as a compact abbreviation for a TFS, supporting clarity and easy modifiability of descriptions (a.k.a. a *type definition*). Furthermore, types can be arranged in a *type hierarchy*, allowing multiple inheritance of information from all supertypes. The next section will give examples.

3 *XTDL*–The Formalism in *SProUT*

XTDL combines two well-known frameworks: typed feature structures and regular expressions.

3.1 The Basis: TDL

XTDL is defined on top of *TDL*, a definition language for TFSs [13] that is used as a descriptive device in several grammar systems (LKB [4], PAGE [29], PET [3]). We use the following fragment of *TDL*, including coreferences.

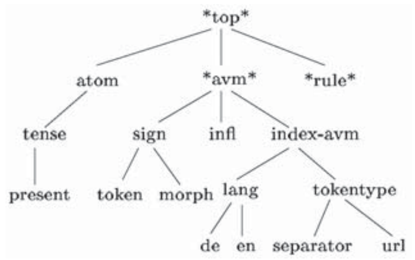
type-def	$\rightarrow$	*type* ":=" *avm* "." |
		type ":<" *type* "." |
		string ":<" *type* "."
type	$\rightarrow$	*identifier*
avm	$\rightarrow$	*term* { "&" *term*} *
term	$\rightarrow$	*type* | *fterm* | *string* | *coref*
fterm	$\rightarrow$	"[" [*attr-val* {"," *attr-val*} *] "]"
attr-val	$\rightarrow$	*identifier avm*
coref	$\rightarrow$	"#"*identifier*

Apart from the integration into the rule definitions, we also employ this fragment in *SProUT* for the establishment of a type hierarchy of linguistic entities. In the example definition below, the *morph* type inherits from *sign* and introduces four morphosyntactically motivated attributes, together with their corresponding values.

```
morph := sign & [POS atom,
                 STEM atom,
                 INFL infl,
                 SEGMENTATION list].
```

POS encodes part-of-speech information, e.g., whether a *morph sign* is a noun, a verb, etc. The STEM feature refers to the main form of a word, e.g., *gut* is the stem of *besser*. The value of INFL is again a feature structure, representing inflectional information. The SEGMENTATION feature is a list-valued feature, encoding a sequence of segments for the compound word.

The next figure depicts a fragment of the type hierarchy used in the example.



3.2 The Regular Extension: XTDL

A rule in *XTDL* is straightforwardly defined as a recognition pattern on the LHS, written as a regular expression, and an output description on the RHS.[1] A named label serves as a handle to the rule. Regular expressions over feature structures describe sequential successions of linguistic signs. We provide a couple of standard operators; see the EBNF below. Concatenation is expressed by consecutive items. Disjunction, Kleene star, Kleene plus, and optionality are represented by the operators |, *, +, and

[1] *XTDL* rules are related to *lexical rules* in UBGs, devices developed for expressing lexical generalizations; see section 3.

?, resp. {*n*} following an expression denotes an *n*-fold repetition, whereas {*m*,*n*} repeats at least *m* times and at most *n* times.

```
rule       → rulename {":>" | ":/"} regexp "->" [avm] [fun-op] "."
rulename   → identifier
regexp     → avm | "@seek(" rulename ")" | "(" regexp ")" |
               regexp {regexp}+ | regexp {"|" regexp}+ |
               regexp {"*" | "+" | "?"} | regexp "{" int [ "," int ] "}"
fun-op     → ", where" coref "=" fun-app {"," coref "=" fun-app}*
fun-app    → identifier "(" term {"," term}* ")"
```

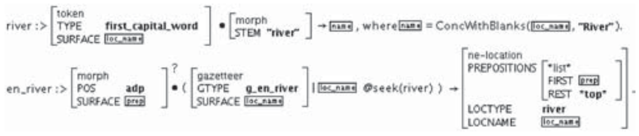
The choice of *TDL* as a basis for *XTDL* has a couple of advantages. TFSs as such provide a rich descriptive language over linguistic structures (as opposed to atomic symbols) and allow for a fine-grained inspection of input items. They represent a generalization over pure atomic symbols. Unifiability as a test criterion (viz., whether a transition is viable) can be seen as a generalization over symbol equality. Coreferences in feature structures express structural identity. Their properties are exploited in two ways. They provide a stronger expressiveness since they create dynamic value assignments while following the transitions in the finite-state automaton, thus exceeding the strict locality of constraints in an atomic symbol approach. Furthermore, coreferences serve as a means of information transport into the output description on the RHS of the rule. Finally, the choice of feature structures as primary citizens of the information domain makes composition of modules simple, since input and output are all of the same abstract data type.[2]

3.3 Two Examples

The *XTDL* grammar rule (1) below may illustrate the the concrete syntax of the formalism. It describes a sequence of morphologically analyzed tokens (of type *morph*). The first TFS matches one or zero items (?) with part-of-speech Determiner. Then, zero or more Adjective items are matched (*). Finally, one or two Noun items ({1,2}) are consumed. The use of a variable (e.g., #c) in different places establishes a coreference (i.e., a pointer) between features. This example enforces, e.g., agreement in case, number, and gender for the matched items. I.e., all adjectives must have compatible values for these features. If the recognition pattern on the LHS successfully matches the input, the description on the RHS creates a feature structure of type *phrase*. The category is coreferent with the category Noun of the right-most token(s) and the agreement features result from the unification of the agreement features of the *morph* tokens.

```
np :> morph & [POS Determiner,
           INFL [CASE #c, NUMBER #n, GENDER #g ]] ?
      (morph & [POS Adjective,
            INFL [CASE #c, NUMBER #n, GENDER #g ]] ) *
      morph & [POS Noun & #cat,
           INFL [CASE #c, NUMBER #n, GENDER #g ]] {1,2}
   –> phrase & [CAT #cat,
           AGR agr & [CASE #c, NUMBER #n, GENDER #g ]].
```

Figure 1 illustrates the conciseness of the formalism (taken from the English named-entity grammar developed by Atsuko Shimada) and addresses the recognition of river names. The first rule matches expressions consisting of an (unknown) capitalized word, followed by a word with stem 'river'. If the LHS applies, the string concatenated by the functional operator ConcWithBlanks then forms the output of the rule. The second rule matches one or more prepositions, followed by either a Gazetteer match (e.g., containing English river names represented by the Gazetteer type g_en_river) or the output of the previous rule (the seek call) bound to the coreference loc_name. The generated output structure of type *ne-location* consists of a list of prepositions, a location type and the (transported) location name. To sum up, the second rule recognizes both unknown river names (via the first rule) and known river names (via a gazetteer entry).



```
river :> [token
          TYPE     first_capital_word
          SURFACE  loc_name] • [morph
                                STEM "river"] → name , where name = ConcWithBlanks(loc_name, "River").

en_river :> [morph
             POS      adp
             SURFACE  prep]? • ( [gazetteer
                                  GTYPE    g_en_river
                                  SURFACE  loc_name] | loc_name @seek(river) ) → [ne-location
                                                                                 PREPOSITIONS [*list*
                                                                                               FIRST prep
                                                                                               REST  *top*]
                                                                                 LOCTYPE      river
                                                                                 LOCNAME      loc_name].
```

Figure 1: A more complex rule combination, involving a functional operator and a seek call, combining tokenizer input, morphology/lexicon and gazetteer lookup. We use the graphical visualization tool from the development environment of SProUT *for depicting the two* XTDL *rules; see also figure 3.*

3.4 Functional Operators, SEEK and a Future Extension

As we already said, *SProUT* differs from other STP systems in using typed feature structures instead of atomic symbols. The system further provides two additional extensions: functional operators and the possibility to call additional rules during the cause of a single rule interpretation (like a call to a subprocedure in a programming language). The latter option even allows a rule to call itself and clearly extends the expressiveness of the formalism, making it context-free (like the related recursive transition networks are). Using the seek operator slightly reduces the efficiency of the grammar, forcing the interpreter to produce new environments for each seek. To improve the efficiency, we introduced an optimizing mechanism for seek calls.

As opposed to the above regular operators |, *, and +, *SProUT* also provides functional operators which are assumed to be the door to the outside world. Figure 3 has already presented the usefulness of a functional operator which concatenates two strings. *SProUT* comes along with a set of predefined functional operators. A new operator must be implemented as a separate Java class and the execution of a specific operator call corresponds to the instantiation of the Java class. Since the *SProUT* interpreter does not know in advance which functional operators are employed in a specific grammar, it dynamically loads a new Java class once at run time during the first call, thanks to Java's reflection API. Not only might a functional operator produce values which are bound and transported via coreferences to other places in a rule. One can even let such an operator act as a predicate, producing only Boolean values which might terminate a rule application, e.g., the production of output via the RHS of a rule.

We are currently implementing a new concept of weaker, unidirectional coreference constraints which are extremely useful under Kleene star (or restricted repetition). The idea here is that the values under such coreferences are collected in a set which is given to the RHS of a rule (we indicate this behavior by using the percent sign in the concrete syntax). Consider, for instance, the *np* rule (1) above and assume that adjectives also

[2] [5] present an integrated architecture for shallow and deep text processing, which further demonstrates the benefits of using TFSs as a representation and interchange format.

have a relation attribute RELN. Our intention is to collect all relations and to have them grouped in a set on the RHS:

```
np :> [POS Det, ...] ([POS Adj, ..., RELN %5])*
      [POS Noun, ...]
      -> [..., RELN %5]
```

A usual coreference marker, however, would enforce that the iterated values under RELN attribute are the same.

4 Architecture

The core of the *SProUT* system consists of four major components: a finite-state machine toolkit, a regular compiler, the *XTDL* interpreter, and a Java typed feature structure package. On top of them, reusable online linguistic processing components have been developed.

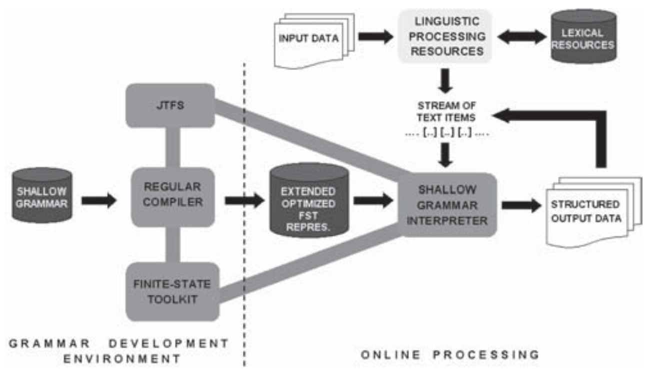


Figure 2: The architecture of SProUT

4.1 Finite-State Machine Toolkit

The FS machine toolkit is a generic toolkit for constructing, combining, and optimizing FS devices [20]. In order to cover all relevant types of FS devices and to allow for a parameterizable weight interpretation, the finite-state machine (FSM) has been chosen as the underlying model. A FSM is a generalization of the more familiar finite-state automaton (FSA), finite-state transducer (FST), and their weighted counterparts [16]. Contrary to weighted FSTs which are tailored to a specific semiring for weight interpretation, the FSMs are more general in that they admit the use of arbitrary real-valued semirings (we use, e.g., the tropical semiring for regular pattern prioritization). All state-of-the-art operations on FSMs are provided, whereas the architecture of the toolkit is mainly based on the tools developed by AT&T [17]. Furthermore, the toolkit is equipped with some new crucial operations relevant to STP, including weighted local extension [26], an efficient algorithm for incremental construction of minimal, deterministic, and acyclic FSAs from a list of words [7], plus a general algorithm for removing ε-moves [17] which has been improved in terms of efficiency.

4.2 Regular Compiler

Since regular expressions are regarded as the adequate level of abstraction for thinking about finite-state languages, we developed a flexible XML-based and Unicode-compatible regular compiler for converting regular patterns into their corresponding compressed finite-state representation [21]. An extendible set of approx. 20 standard regular operators is provided. The input data can be interpreted either as a scanner definition (e.g., token types) or general regular expressions (e.g., regular expressions over TFSs).

Both the definition and configuration of the transformation process is done via XML which allows for straightforward extensions. Grammarians may even flexibly bias the process of merging and optimizing FS devices. For instance, the way in which ambiguities are handled is triggered by the user via a choice of two alternative options. In the first one, potential ambiguities are resolved by assigning weights to the patterns which represent their priorities, and applying the tropical semiring in the process of merging them into a single FS device (e.g., in the tokenizer of *SProUT*). The second option is to preserve all ambiguities by introducing appropriate final emissions, representing pattern identifiers in the corresponding FS devices (e.g., in shallow grammars in *SProUT*). Further subtleties such as the choice of the minimization algorithm or definition of filters which convert existing resources spread over external databases into FS representation can be specified by the user.

The compilation of *XTDL* grammars is straightforward. The TFSs of the production part in the LHS of each rule are replaced by symbols representing references to these structures, since FSM arcs may be only labeled with symbolic values. Subsequently, all such modified LHS are transformed into a corresponding FS network. Note that through the use of final emissions mentioned above, an association link between LHSs with their corresponding RHSs and original rules, is preserved.

Since TFS annotations on arcs of the finite-state networks usually do not allow for determination and minimization of such networks under TFS equivalence, a handful of methods going beyond standard finite-state techniques have been developed to alleviate this problem [12].

4.3 XTDL Interpreter

The challenge for the *SProUT* interpreter is to combine regular expression matching with unification of TFSs. Since the regular operators such as Kleene star can not be expressed by a TFS, the interpreter algorithm is faced with the problem of mapping a regular expression to a corresponding sequence of TFS, so that the coreference information among the elements in a rule is preserved. The solution is to separate the matching of regular patterns using unifiability (LHS of rules) from the construction of the output structure through unification (RHS). The positive side effect is that the fast matching step filters the potential candidates for the space-consuming unification. After a compatible pattern is identified, the sequence of input TFSs is embedded (encoded as a list) into a new TFS.

Subsequently, a rule TFS with an instantiated LHS pattern is constructed. A TFS representation of a rule contains the two attributes IN and OUT. In contrast to the IN value in the matched input TFS representation, the IN value of the rule contains coreference information. The value of OUT is the TFS definition of the RHS of the rule. Given the input TFS and the uninstantiated rule TFS, the unification of the two structures yields the final output result.

The use of coreferences between the LHS and the RHS of a *SProUT* rule shares great similarities with lexical rules in PATR-II [28] and HPSG [24]. The technique of embedding an instantiated LHS pattern and a RHS via the metafeatures IN and OUT also reminds us of the PATR-II system.

The current implementation employs a longest match strategy. In case of match ambiguities, the result is a disjunction of RHSs. Since the output of the interpreter are again TFSs, the result can be used as input for further (higher-level) linguistic processing components. In this way, *SProUT* supports cascaded architectures straightforwardly (see section 5).

4.4 Typed Feature Structure Package

The JTFS package is a Java implementation of TFSs. JTFS reads in a binary representation of a typed UBG, including type hierarchy and lexicon, and builds up the object in main memory.[3] The destructive lazy-copying unifier in JTFS is an optimized variant of [8], together with an efficient type unification operation (bit vector bitwise AND plus caching). JTFS supports a dynamic extension of the type hierarchy at run time in order to allow for the incorporation of unknown words. Other operations, such as subsumption, unifiability testing, deep copying, path selection, feature iteration, and different printers are available.

Since the unifiability testing in *SProUT* is crucial for the efficiency of the whole system, we have developed a further imperfect, but extremely fast unifiability test that does not require the space of the standard test to store the effects of the unification. The test is imperfect in that there are very rare combinations of feature structures which are assumed to be unifiable, but which are not. Such combinations are detected later in the construction of the RHS of a rule (see interpreter section above) when performing standard unification. The important part, however, is that almost all unifiability tests during grammar interpretation fail and for these cases the fast test delivers a correct answer.

4.5 Processing Components

An application of a compiled grammar to a given text consists of two steps. Firstly, the input text is processed via a stream of linguistic processing components specified explicitly by the user. These components produce several streams of so called text items which constitute the input for the *XTDL* interpreter described earlier. Currently the pool of linguistic processing resources contains a tokenizer, a gazetteer, a morphology component, and a reference matcher. The tokenizer maps character sequences of the input text into word-like units called tokens. Many IE tasks may be solved almost solely via the application of a tokenizer. Hence, this component was defined for performing fine-grained multiple token classification. Each token is firstly classified according to the main token type and secondly, depending on its main type, it undergoes additional domain and language specific subclassification.

Since we aim at defining clear-cut components of linguistic analysis, the context information is disregarded during token classification. Therefore, sentence boundary detection constitutes a stand-alone module.

The task of the gazetteer is the recognition of full names (e.g., locations, organizations) and keywords (e.g., company designators) based on static lexicons. The gazetteer entries may be associated with a list of arbitrary attribute-value pairs which strongly supports text normalization.

The morphology unit provides lexical resources for English, German, Dutch, French, Italian, and Spanish, which were compiled from the full form lexicons of MMorph [19].[4] Additionally, this module is equipped with an online shallow compound recognition for German and Dutch. Considering Slavic languages, a component for Czech [9] and Polish [25] has been integrated. For Asian languages, we use Chasen [1] for Japanese and Shanxi [15] for Chinese.

Finally, the task of the reference matcher is to find identity relations between entities previously recognized in the text (e.g., variants of the same named entity). Note that this component runs after grammar interpretation. It takes as input the output of the *XTDL* interpreter potentially containing user-defined information on variant construction for certain entity classes and performs an additional pass through the text for identification of previously unrecognized entities.

Easy composition of linguistic processing resources is facilitated in *SProUT*, since input and output data are uniformly represented as TFSs. In the current system, components are arranged in a strictly sequential fashion. In order to overcome this inflexible behavior, the system description language *SDL* has been developed [11] which allows the construction of a concrete system instance by means of a regular expression over module names. *SDL* provides operators for expressing concatenation, (self-)iteration, and parallel execution of modules. Given a declarative system specification, *SDL* finally compiles an executable Java program, realizing the intended behavior of the original system specification.

5 Applications

IE systems are becoming commercially viable in supporting diverse information discovery and management tasks. The *SProUT* platform has been adopted as the core IE component in several EU-funded and industrial projects, supporting tasks like content extraction and acquisition for text/data mining, dynamic hyperlinking, machine translation, and text summarization. These applications yield valuable feedback for further improvements and extensions of the system.

5.1 Integrating Information Extraction and Automatic Hyperlinking.

ExtraLink [2] is a novel information system combing IE technology and automatic hyperlinking. Automatic hyperlinking is a maturing technology designed to interrelate pieces of information, using ontologies to define relationships between concepts. Semantic concepts identified by the *SProUT* named-entity recognition component are mapped onto a domain ontology that relates concepts to a selection of hyperlinks, which are directly visualized on demand using a standard web browser. This way, the user can, while reading a text, immediately link up textual information to the Internet or to any other document base without accessing a search engine. *ExtraLink* showcases the extraction of relevant concepts from German texts in the tourism domain, offering the direct connection to associated web documents on demand.

5.2 Multilingual Information Extraction for AIR FOReCast in Europe.

The EU-funded AIRFORCE project aims at developing ideas and components which support building a database of European events and trends, helping to forecast air traffic. AIRFORCE adopts *SProUT* for building up domain-specific named entity

[3] The binary grammar representation is produced by the *flop* preprocessor of the PET system [3].

[4] [14] describes an offline compaction method that removes both redundancies and spurious ambiguities from MMorph. The described technique has drastically increased the efficiency of *SProUT* since it shrinks the size of the lexicon and comes up with fewer readings for a morphological form.

and relation extraction grammars and extracting relations automatically from official travel warnings, published regularly in the Internet by the ministries for foreign affairs of France, Germany, and the UK. The results of the extraction process are used to fill a database. *SProUT* has been extended to meet the specific needs of delivering a language-neutral output for English, French, or German input texts. A shared type hierarchy, a feature-enhanced gazetteer resource, and generic techniques of merging chunk analyses into larger results are major reusable results.

5.3 Multilingual IE for Machine Translation and Text Summarization.

The EU-funded MEMPHIS project has developed a platform for cross-lingual premium content services, targeting mainly portable thin clients, like mobile phones, PDAs, etc. The core of the system is a cross-lingual transformation layer, integrating cross-lingual information extraction and summarization of source documents, translation to the customers' target languages, a crosslingual knowledge management for extracted information based on an application's domain ontology as well as multi-lingual generation of documents according to the restrictions and requirements of the various target devices for distribution. *SProUT* is used on the one hand as a document processing engine for tokenization, morphological analysis, and named-entity recognition. On the other hand, the named-entity recognition has also been integrated into the machine translation and the text summarization system for boosting their performance.

5.4 Information Extraction for Polish.

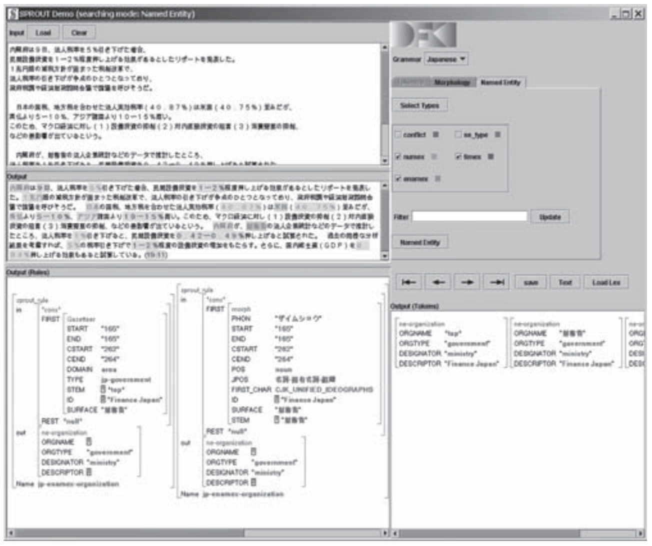
An attempt in applying *SProUT* in the process of constructing an Information Extraction engine for Polish and adopting it to the processing of Slavic languages are reported in [22]. The IE tasks focus on the identification of typical named entities from financial texts and on extraction of data about pathological changes from a medical corpus containing descriptions of mammographical examinations.

5.5 Opinion Mining.

The ArgoServer system, developed by the Italian company Celi, analyzes on a daily basis forums and newsgroups on different car manufacturers in order to retrieve interesting messages and trends. *SProUT* is applied here to handle the information extraction task. The extracted opinions are input to statistical postprocessing, yielding, e.g., the total number of comments (or attitudes) expressed by the forum/newsgroup users in the monitored period.

5.6 Hybrid Deep and Shallow Methods for Knowledge-Intensive Information Extraction

In the DeepThought project, English, German, and Japanese named entity recognition of *SProUT* is employed in a hybrid architecture integrating deep and shallow natural language processing components (see http://www.project-deepthought.net). Prototype application domains are precise information extraction for business intelligence, e-mail response management for customer relationship management, and creativity support for document production and collective brainstorming. Here, the *SProUT* XML output is converted to the semantic formalism RMRS (robust minimal recursion semantics) through XSLT stylesheets [27].



```
sprout_rule                              sprout_rule
in   "cons"                              in   "cons"
     FIRST Gazetteer                          FIRST morph
           START     "165"                          PHON       "ザイムショウ"
           END       "165"                          START      "165"
           CSTART    "263"                          END        "165"
           CEND      "264"                          CSTART     "263"
           DOMAIN    area                           CEND       "264"
           TYPE      jp-government                  POS        noun
           STEM      ▯ "top"                        JPOS       [illegible]
           ID        ▯ "Finance Japan"              FIRST_CHAR CJK_UNIFIED_IDEOGRAPHS
           SURFACE   "[illegible]"                  ID         ▯ "Finance Japan"
     REST "null"                                    SURFACE    "[illegible]"
out  ne-organization                                STEM       ▯ "[illegible]"
     ORGNAME    ▯                             REST "null"
     ORGTYPE    "government"             out  ne-organization
     DESIGNATOR "ministry"                    ORGNAME    ▯
     DESCRIPTOR ▯                             ORGTYPE    "government"
Name jp-enamex-organization                   DESIGNATOR "ministry"
                                              DESCRIPTOR ▯
                                         Name jp-enamex-organization
```

```
ne-organization                      ne-organization
ORGNAME    "top"                     ORGNAME    "[illegible]"
ORGTYPE    "government"              ORGTYPE    "government"
DESIGNATOR "ministry"                DESIGNATOR "ministry"
DESCRIPTOR "Finance Japan"           DESCRIPTOR "Finance Japan"
```

Figure 3: The SProUT grammar development environment.

Acknowledgments

We like to thank our former colleagues Markus Becker and Oliver Scherf who have contributed several ideas presented in this article. Thanks are also due to Stephan Busemann and Hans Uszkoreit for their moral support. We are grateful to the *SProUT* grammar developers and users who have provided us with valuable and constructive comments, helping us to improve the overall approach and performance of the system. Finally, we like to thank our reviewer for useful comments. The work reported in this article has been partially funded by the BMBF under grant nos. 01 IW 002 (Whiteboard), 01 IN A01 (Collate), 01 IW C02 (Quetal) and by the EU under grant nos. IST-12179 (Airforce), IST-2000-25045 (Memphis), IST-2001-37836 (DeepThought), and by additional non-financed personal effort of the authors.

References

[1] Masayuki Asahara and Yuji Matsumoto. Extended models and tools for high-performance part-of-speech tagger. In *Proceedings of COLING 2000*, 2000.

[2] Stephan Busemann, Witold Drożdżyński, Hans-Ulrich Krieger, Jakub Piskorski, Ulrich Schäfer, Hans Uszkoreit, and Feiyu Xu. Integrating information extraction and automatic hyperlinking. In *Proceedings of the Interactive Posters/Demonstration at ACL-03*, pages 117–120, 2003.

[3] Ulrich Callmeier. PET – a platform for experimentation with efficient HPSG processing. *Natural Language Engineering*, 6(1):99–107, 2000.

[4] Ann Copestake. The (new) LKB system. CSLI, Stanford University, 1999.

[5] Berthold Crysmann, Anette Frank, Bernd Kiefer, Stefan Müller, Günter Neumann, Jakub Piskorski, Ulrich Schäfer, Melanie Siegel, Hans Uszkoreit, Feiyu Xu, Markus Becker, and Hans-Ulrich Krieger. An integrated architecture for shallow and deep processing. In ˜*Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, ACL-2002*, pages 441–448, 2002.

[6] Hamish Cunningham, Diana Maynard, and Valentin Tablan. JAPE: a Java annotation patterns engine (second edition). Research memorandum cs-00-10, Department of Computer Science, University of Sheffield, November 2000.

[7] Jan Daciuk. *Incremental Construction of Finite-State Automata and Transducers, and their Use in the Natural Language Processing*. PhD thesis, Technical University of Gdask, Gdask, Poland, 1998.

[8] Martin Emele. Unification with lazy non-redundant copying. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, pages 323–330, 1991.

[9] Jan Hajič. Disambiguation of rich inflection (computational morphology of Czech), 2001. Karolinum, Charles University Press, Prague.

[10] J. Hobbs, D. Appelt, J. Bear, D. Israel, M. Kameyama, M. Stickel, and M. Tyson. Fastus: A cascaded finite-state transducer for extracting information from natural-language text. In E. Roche and Y. Schabes, editors, *Finite State Devices for Natural Language Processing*. MIT Press, 1997.

[11] Hans-Ulrich Krieger. *SDL* – a description language for specifying NLP systems. In *Proceedings of the 3rd AMAST Workshop on Algebraic Methods in Language Processing, AMiLP-3*, 2003.

[12] Hans-Ulrich Krieger and Jakub Piskorski. Speed-up methods for complex annotated finite state grammars. DFKI Report, 2004.

[13] Hans-Ulrich Krieger and Ulrich Schäfer. *TDL* – a type description language for constraint-based grammars. In *Proceedings of the 15th International Conference on Computational Linguistics, COLING-94*, pages 893–899, 1994.

[14] Hans-Ulrich Krieger and Feiyu Xu. A type-driven method for compacting MMorph resources. In *Proceedings of RANLP 2003*, pages 220–224, 2003.

[15] Kaiying Liu. Research of automatic chinese word segmentation. In *International Workshop on Innovative Language Technology and Chinese Information Processing (ILT&CIP-2001)*, 2001.

[16] Mehryar Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 2(23):269–311, 1997.

[17] Mehryar Mohri, Fernando Pereira, and Michael Riley. A rational design for a weighted finite-state transducer library. Technical report, AT&T Labs - Research, 1996.

[18] G. Neumann, R. Backofen, J. Baur, M. Becker, and C. Braun. An information extraction core system for real world German text processing. In *th International Conference of Applied Natural Language*, pages 208–215, 1997.

[19] Dominique Petitpierre and Graham Russell. MMORPH – The Multext Morphology Program, 1995. Multext Deliverable 2.3.1. ISSCO, University of Geneva.

[20] Jakub Piskorski. The DFKI finite-state machine toolkit. Research Report RR-02-04, German Research Center for Artificial Intelligence (DFKI), 2002.

[21] Jakub Piskorski, Witold Drożdżyński, Feiyu Xu, and Oliver Scherf. A flexible XML-based regular compiler for creation and converting linguistic resources. In *Proceedings of the 3rd International Conference on Language Resources and Evaluation, LREC-2002*, 2002.

[22] Jakub Piskorski, Petr Homola, Małgorzata Marciniak, Agnieszka Mykowiecka, Adam Przepiórkowski, and Marcin Woliński. Information extraction for polish using the SProUT platform. In *Proceedings of Intelligent Information Systems*, 2004.

[23] Jakub Piskorski and Günter Neumann. An intelligent text extraction and navigation system. In *Proceedings of the 6th International Conference on Computer-Assisted Information Retrieval, RIAO-2000*, 2000.

[24] Carl Pollard and Ivan A. Sag. *Head-Driven Phrase Structure Grammar*. Studies in Contemporary Linguistics. University of Chicago Press, Chicago, 1994.

[25] Adam Przepiórkowski and Marcin Woliński. A flexemic tagset for Polish. In *Proceedings of the Workshop on Morphological Processing of Slavic Languages, EACL*, 2003.

[26] Emmanuel Roche and Yves Schabes. Deterministic part-of-speech tagging with finite state transducers. *Computational Linguistics*, 21(2):227–253, 1995.

[27] Ulrich Schäfer. WHAT: An XSLT-based infrastructurefor the integration of natural language processing components. In *Proceedings of the HLT-NAACL Workshop on the Software Engineering and Architecture of Language Technology Systems, SEALTS*, pages 9–16, 2003.

[28] Stuart Shieber, Hans Uszkoreit, Fernando Pereira, Jane Robinson, and Mabry Tyson. The formalism and implementation of PATR-II. In Barbara J. Grosz and Mark E. Stickel, editors, *Research on Interactive Acquisition and Use of Knowledge*, pages 39–79. AI Center, SRI International, Menlo Park, Cal., November 1983.

[29] Hans Uszkoreit, Rolf Backofen, Stephan Busemann, Abdel Kader Diagne, Elizabeth A. Hinkelman, Walter Kasper, Bernd Kiefer, Hans-Ulrich Krieger, Klaus Netter, Günter Neumann, Stephan Oepen, and Stephen P. Spackman. DISCO – an HPSG-based NLP system and its application for appointment scheduling. In *Proceedings of COLING-94*, pages 436–440, 1994.

[30] Gertjan van Noord and Dale Gerdemann. Finite state transducers with predicates and identity. *Grammars*, 4(3):263–286, 2001. Kluwer.

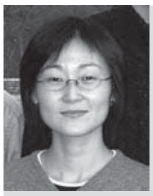
Contact
Witold Drożdżyński, Hans-Ulrich Krieger,
Jakub Piskorski, Ulrich Schäfer, Feiyu Xu
German Research Center for Artificial Intelligence (DFKI)
Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany
sprout@dfki.de

Witold Drożdżyński graduated at the Poznan University of Economics with a masters degree in economics and computer science. Since 2000, he is a software engineer at DFKI. He has experience in database systems, WEB applications and graphical user interface, and has developed applications in Java and C++. Before starting at DFKI, he has worked for industrial projects in a private company.

Dr. Hans-Ulrich Krieger is a Senior Researcher at DFKI, studied computer science and physics at the RWTH in Aachen, and received a PhD in Computer Science from Saarland University in 1995. His global research focuses on linguistic formalisms, their efficient implementation, and their mathematical foundations. Krieger's latest work concerns the compilation of constraint-based formalisms into weaker frameworks and the integration of deep and shallow processing.

Dr. Jakub Piskorski received his M.Sc. in computer science from The University of Saarbrücken, Germany, in 1994 and Ph.D from the Polish Academy of Sciences in Warsaw, in 2002. He is a member of the Language Technology Lab of DFKI since 1998, and previously worked at the University of Economics in Poznan, Poland. His areas of interest are centered around Finite-State Technology, Shallow Text Processing, Information Extraction, Text Mining, and efficient application-oriented NLP solutions.

Ulrich Schäfer studied computer science and computational linguistics at Saarland University. He received his diploma in 1995. Thereafter, he worked as software developer and consultant in the fields of multilingual document management, directory services, and electronic data interchange. Since 2000, he is Senior Software Engineer in the DFKI Language Technology Lab. His current focus is on architectures for the integration of deep and shallow natural language processing components and question answering systems.



Feiyu Xu has been working as at first researcher and then as senior software engineer at DFKI LT-lab, since she finished her studies in the Computational Linguistics at the University of the Saarland in 1998. She has been involved in different international and national projects concerning information management systems. Her main research areas are multilingual/cross-lingual information retrieval, information extraction, text data mining, and question answering.