Centre of Natural Language Processing
Faculty of Informatics
Masaryk University
Brno

Computer Processing of Czech Syntax and Semantics

Aleš Horák

Brno, 2008

Aleš Horák
Faculty of Informatics, Masaryk University
Centre of Natural Language Processing (NLP Centre)
Botanická 68a
CZ-602 00 Brno, Czech Republic
E-mail: hales@fi.muni.cz

Reviewed by Karel Pala, Masaryk University, Czech Republic

ISBN: 978-80-7399-375-7

Preface

This book presents the results of research obtained during the course of a number of natural language processing projects that were led by Aleš Horák in the Centre of Natural Language Processing (aka NLP Centre or NLP laboratory), Faculty of Informatics, Masaryk University in Brno. As such, the presented results are based on the team work of researchers as well as students who were directly participating in the projects.

The whole text offers a survey of sophisticated research methods concentrating on two complex levels of natural language (NL) processing, namely those of syntax and semantics. However, we do not try to cover all approaches in these areas – we focus on rule-based introspective methods with an encapsulation of empirical paradigms in the form of Figures of Merits (FOMs) of particular syntactic and semantic phenomena.

As a basis for many NLP research projects, we have developed a number of advanced natural language processing tools and language resources. In the second chapter, we offer a detailed description covering three years of the development of VerbaLex, a large lexicon of Czech verb valencies in the form of complex valency frames. This part is then followed by the presentation of specific software developed for working with this as well as other language resources. These tools are VisDic, DEBVisDic, DEBDict, PRALED and others. These tools are used by project teams all over the world.

The next chapter presents the latest development of the syntactic analyser `synt` that has been under development in the NLP Centre for several years. Besides the comprehensive description of `synt` inside and formats used, we also provide a comparison with several other natural language parsers, in which we show that the `synt` qualities are at least comparable to the best current parsers.

The fourth chapter outlines the advances made in the Normal Translation Algorithm (NTA) from [Hor02]. It describes the methods and techniques aimed at an automatic translation from a NL sentence to its meaning expressed as a construction in Transparent Intensional Logic. The description is not complete, yet, but we have concentrated on selected phenomena where we offer sample solutions or even prototypical implementations.

The last chapter gives details of a project that concentrates on intelligent methods for increasing the reliability of electrical networks. One its part involves the development of a human-machine communication framework for dialogues about the specific knowledge domain of electrical power systems (EPS). The task of another project part is the development of a multi-agent system for representing the EPS processes. These allow simulating different configurations of an EPS setup with an automatic computation of the economic aspects of the system failures.

Acknowledgements

For more than 10 years, all research in the NLP Centre has been driven and inspired by its head, Karel Pala. For this reason above all others he is the first person I would like to thank for his invaluable help and support. In addition, my thanks for cooperation on this work go to the following NLP Centre members (in alphabetical order): Pavel Cenek, Andrej Gardoň, Jiří Golembiovský, Dana Hlaváčková, Vladimír Kadlec, Vojtěch Kovář, Martin Kudlej, Petr Pelán, Martin Povolný, Miroslav Prýmek, Adam Rambousek, Pavel Rychlý, Lukáš Svoboda, Marek Veber and Radek Vykydal, as well as co-authors of project publications from other research teams: Marie Duží, Tomáš Holan, Pavel Materna, Pavel Smrž, and Piek Vossen.

The financial support of this work was provided by the Faculty of Informatics, Masaryk University, Brno and by the following grant projects:

- the EU project Balkanet IST-2000-29388,
- the Czech Science Foundation projects 201/05/2781 and 405/03/0913,
- the projects 1ET100300414, 1ET100300419 and 1ET200610406 of the Grant Agency of the Academy of Sciences of CR,
- the Ministry of Education of the Czech Republic within the Re-

search Intent CEZ:J07/98:143300003,

- the Ministry of Education of the Czech Republic within the Center of basic research LC536, and
- the Ministry of Education of the Czech Republic in the National Research Programme II project 2C06009.

Contents

Chapter 1

Introduction

The Natural Language Processing Laboratory was established at the Faculty of Informatics, Masaryk University in 1997 within the project VS97028 of the Ministry of Education of the Czech Republic. The founding members of the laboratory at that time were Karel Pala, Aleš Horák, Pavel Rychlý and Pavel Smrž. In 2005 the NLP laboratory was renamed to the Centre of Natural Language Processing. Since its beginnings the NLP laboratory or the NLP Centre has remained true to the essential notion of its *raison d'être*: – the NLP Centre is a place where about twenty researchers and dozens of undergraduate and postgraduate students work on research tasks from the exciting domain of the computational processing of written and spoken natural language. In this text, we present the details of four of the projects in which the NLP Centre has been engaged over last five years. What all these projects have in common is the fact that the author of this text is the leading person of the project work.

As a first project we introduce new language resources and new language processing tools developed in the NLP Centre. The most important of the resources described is a new lexicon of complex valency frames of Czech, named VerbaLex. The lexicon includes all the usual verb valency features plus additional relevant information such as verb aspect, verb synonymity, types of use and semantic verb classes based on the VerbNet project. An important property of VerbaLex as far as the computer processing is concerned, is its close relationship with the widely exploited English and Czech WordNet semantic networks.

We also present new tools based on client/server XML database system called DEB II. Thanks to the versatility of the XML format used, this system enables us to cover various applications, namely the management of the electronic readable dictionaries, WordNet-like lexical databases as well as ontologies for Semantic Web applications. Considerable attention is paid to the inner workings of the DEB II framework as well as to particular DEB client tools especially to WordNet development tools VisDic and DEBVisDic, which represents a well-designed and developed system for lexical database editing and is currently employed in many national WordNet building projects. We discuss the basic features of the tools as well as more elaborate functions that facilitate linguistic work in multilingual environments. We argue for the benefits the new DEB II platform brings to WordNet editing and to XML databases in general.

In the following text, the main features and assets of the DEB II dictionary writing platform are outlined, and the implementation strategies of both server and client parts of the platform are characterized. We also pay attention to the process of merging lexical data, particularly Czech WordNet with VerbaLex, the list of Czech valency frames, developed separately in XML format. We show an example of the merge which also indicates how DEBVisDic can serve as a means for such a kind of integration. We also point out that this type of merge can be extended to other languages as it was done with Bulgarian and Romanian in the Balkanet project.

The reader will also find here an overview of the current state of other DEB II applications that include:

- PRALED – a client for building the Czech Lexical Database,
- DEBDict – a browser for parallel viewing of several electronic dictionaries,
- Cornetto – an augmented WordNet-like lexical database system,
- DEB CPA browser and editor – a client for building a database of verb patterns as they derive from corpora (called *corpus patterns*),
- DEB TEDI tool – an application for building specialized terminological dictionaries.

For each of these DEB clients we describe and demonstrate their main features and functionality.

The third chapter presents a survey of the latest development of the Czech sentence parsing system `synt`. The presented system uses the meta-grammar formalism, which allows us to define the grammar with a sustainable number of meta-rules. At the same time, these meta-rules can be automatically translated into rules for efficient and fast head-driven chart parsing and supplemented with an evaluation of additional contextual constraints. In general, the system represents a rule-based approach to syntactic analysis, however, within the parsing process `synt` uses empirical parameters of the input lexical items in the form of so-called Figures of Merits (FOMs). In this respect, the system may be viewed as a combination of the rule-based approach and stochastic approaches involved within most of the Czech language parsing systems developed by members of the research group of Jan Hajič [HH07].

The text includes a comprehensive description of the meta-grammar constructs in `synt` as well as actual running times of the system tested on corpus data. The lexicographer's environment, the so-called Grammar Development Workbench (GDW), is integrated with `synt` into one system that allows a team of experts (computational linguists as well as programmers) to cooperate on the development of a grammar covering all frequent Czech language phenomena.

Besides the description of the `synt` system, we illustrate the process of the meta-grammar development. One of the first phases is formed by constructing corpus data for testing. We demonstrate the exploitation of the corpus on testing a method for detecting the Best Analysis Selection with the results of testing the synt analysis on a Czech corpus. The section about the Best Analysis Selection discusses methods that enhance the algorithm determining "the best" parsing tree from the output of natural language syntactic analysis. It presents a method for pruning redundant parse trees based on the information obtained from a dependency tree-bank corpus.

The VerbaLex valency lexicon from the second chapter is exploited in the syntactic parsing process. The description of the verb frame extraction algorithm and the measured results of running its implementation on a newspaper corpus is displayed as one of the language specific features used in the tree ranking algorithm, which is a crucial part of the `synt` mode of analysis.

The effectiveness of the enhanced parser is demonstrated by results of two inter-system parser comparison experiments. The first tests were run on the standard evaluation grammars, namely ATIS, CT and PT, where the `synt` system outperforms the referential implementations. The second experiment compared the effectiveness of real text parsers of Czech based on completely different approaches – stochastic parsers that provide dependency trees as their outputs, and the meta-grammar parser that generates a resulting chart structure representing a packed forest of phrasal derivation trees. We describe and formulate the main questions and problems accompanying such experiments, try to offer answers to these questions and finally display factual results of the tests as measured on 10 thousand Czech sentences.

In the fourth chapter, we describe the extended type hierarchy of the Transparent Intensional Logic (TIL [Tic04]) as a higher order logic theory. We also present the basic ideas of TIL constructions as a suitable natural language knowledge representation.

TIL is a logic system, designed for representing the meaning of natural language expressions. The system is built on a typed λ-calculus logic with a hierarchy of types. It was created as a parallel to Montague's logic [Mon74], however TIL is more capable of describing natural language semantics while retaining the simplicity of the basic idea. Moreover, the inference rules for TIL are well defined, thus enabling us to use constructions as an instrument for representing sentence meaning in knowledge base systems. The connection between a construction and the constructed object is fact-independent and is driven by the mechanism of typed λ-calculus. Constructions carry information about relations between the elementary parts of language expression objects.

TIL was introduced by Pavel Tichý [Tic88] with the purpose of overcoming paradoxes arising from other modern logical systems (first order predicate logic as well as intensional logics [Mon74]). A short summary of the advantages of TIL over Montague's dynamic logic can be found in [Hor02]. TIL is well suited to handle the difficult language phenomena such as temporal relations, (hyper)intensionality and propositional attitudes. The techniques described in this text are part of the long-term development of the Normal Translation Algorithm aimed at the automatic translation of natural language sentences into TIL constructions.

We describe methods for exploiting the VerbaLex valency frames lexicon in relation to the transparent intensional logic. We examine the relations between complex valency frames (CVFs) and TIL constructions of predicate-argument structures and discuss the procedure of automatic acquisition of the verbal object constructions.

At the end of the chapter, the design of a newly developed Dolphin system for the effective implementation of a knowledge base and basic question answering based on the transparent intensional logic is explained. We will introduce the database acting as a knowledge base for inference in TIL. The time aspect of the truth value of propositions will be included and the basic "thinking" capabilities of the Dolphin system will be exemplified.

We also present an experiment in which the purely logically oriented type system of TIL is compared with the property-based types of the Easel [Fis99] world fact simulation language. The discussion is also oriented towards the possibility of applications combining the two approaches.

The last of the projects we present deals with the role of biologically motivated emergent systems and intelligent agents in the simulation of electric power networks. The main aim is to provide a platform for analyzing the databases of failures of power systems in the Czech Republic (and a part of Slovakia) and to point out where the potential weak points are. The developed system, called Rice, is designed for simulating electrical power system processes. The system is based on the multi-agent approach, which allows a unique versatility of the design and development of the particular power system network. The applications of the system aim at off-line analysis and prediction of power system failure. The communication among the defined agents is based on standards in multi-agent systems – the communication protocols CORBA (Common Object Request Broker Architecture) and KQML (Knowledge Query and Manipulation Language). The system itself economizes the open source implementation of the protocols.

The whole system is able to perform an active simulation of the energy flow in a power system and its visualization. We take into account a possible future replacement of any particular agent with an on-line power equipment monitoring facility (ad-hoc sensors), which allows the whole power system in real time to be monitored..

The latest developed features in the Rice system allow users to capture the behaviour of dynamic emergent networks. To fulfil this requirement, it has to cope with the complexity of local and global changes in the network characteristics including such basic ones as the network topology is. The text also includes several examples of typical power network components and the definitions of their behaviour in the emergent network environment.

We further present an explication of the natural language processing tasks of the NLP Centre in the design and development of a natural language dialogue interface for querying large databases with temporal data about electrical power network failures. The implementation of such a dialogue interface includes the creation and preparation of several auxiliary resources that are required for natural language processing of texts over this specific domain. In this text, we describe the process and statistical results of the creation of a corpus of electrical power networks texts consisting of more than 1 million positions. We also offer preliminary results of the syntactic analysis of the specialized corpus data and describe the problems of morphological and syntactic analysis of such domain specific texts.

Chapter 2

New Language Resources and Tools

Each long term natural language processing (NLP) project needs to work with a firm basis of representative language resources and appropriate tools for their processing. The NLP Centre at the Masaryk University in Brno is taking part in research in all areas of the NLP field with the "handicap" of concentrating on the Czech language, in contrast to the most frequent languages like English, German or French. The situation with the Czech language resources is, however, getting better during last years with the valuable corpus resources at the beginning (DESAM [PRS97], the Czech National Corpus [CNC06, KKK00] or the Prague Dependency Treebank [Haj04a]) and the consecutive specialized resources including the verb frame lexicons VALLEX [ŽL04] and VerbaLex (see the Section 2.1).

In the first section of this chapter, we summarize the three years of building the new VerbaLex lexicon of Czech verb frames containing more than 10 000 verbs. The features of the lexicon are designed to bring important semantic information to computer processing of predicate constructions in running texts in the form of *complex valency frames* (CVFs). The most notable attributes of CVFs include synset (synonymical set) organization, two-level semantic labels with linkage to the Princeton Word-

Net and EuroWordNet[1] hierarchy and surface verb frame patterns used for automatic syntactic analysis.

In the area of lexical databases, ontologies and common sense knowledge resources, the Princeton WordNet [Mil90] became one of the most popular ones. It is currently used in many areas of natural language processing such as information retrieval, automatic summarization, document categorization, question answering, machine translation etc. To integrate into the applications, many researchers work with the Princeton database and transform data to their own proprietary formats.

The Princeton team also developed a data browser for WordNet which could be downloaded together with English data from the WordNet project web page [WN07] both for Windows and UNIX platform. Currently, this browser is replaced with a purely web access application. No WordNet editing tools are provided as the only instruments for majority of the lexicographic work in Princeton are standard text editors. The consistency of data is not therefore checked during the editing process itself, it is postponed to later phases.

Year by year the number of Princeton WordNet derivatives and WordNet-inspired initiatives increased. In 1998–1999 the EU project EuroWordNet 1 and 2 [Eur99] took place, in which multilingual approach dominated and WordNets for 8 European languages, particularly for English, Dutch, Italian, Spanish, French, German, Czech and Estonian, were developed. The Interlingual Index (ILI), the Top Ontology, set of Base Concepts and a set of Internal Language Relations were introduced as well [Vos98]. These changes also led to the design and development of a new database engine for EuroWordNet and it resulted in the editing and browsing tool called Polaris [Lou98].

In 2001–2004 the EU project Balkanet [Bal04] was launched which can be viewed as a continuation of the EuroWordNet project. It was conceived as multilingual, as well, and within its framework WordNets for 6 languages were developed or augmented, particularly for Greek, Turkish, Romanian, Bulgarian, Serbian and Czech. Before Balkanet had started it was already obvious that the Polaris tool had no future because its development had been closed and as a licensed software product (by Lernout and Hauspie) it had been rather expensive for most of the research institutions involved (typically universities). Moreover, the system

[1]see further text for details

had been provided only for MS Windows platform. That is why a specialized open source software system VisDic has been developed by MU NLP Centre during the work within the Balkanet project. VisDic was designed and implemented as a highly configurable multiplatform tool with easy to use interface for working not only with WordNets, but with general dictionaries using a variant of XML schema for the dictionary entries. We present the details of this tool in the Section 2.2.

In June 2000, the Global WordNet Association (GWA, see the web site [GWA07]) was established by Piek Vossen and Christiane Fellbaum. The purpose of this association is to "provide a platform for discussing, sharing and connecting WordNets for all languages in the world." One of the most important actions of GWA is the Global WordNet Conference (GWC) that is being held every two years on different places all over the world. The second GWC was organized by the MU NLP Centre in Brno and the NLP Centre members are actively participating in GWA plans and activities. A new idea that was born during the third GWC in Korea is called the Global WordNet Grid with the purpose of providing a free network of smaller (at the beginning) WordNets linked together through Interlingual Ontology (as opposed to Interlingual Index from EuroWordNet). The Grid preparation is currently just starting and the MU NLP Centre is developing its software background.

The growing need to handle various lexical resources that take the form of dictionaries, semantic networks, ontologies, valency lexicons, or FrameNets is the cause why researchers seek for software systems that are able to store dictionary-like data using XML as the core element. Many dictionary publishing houses operate large systems with the complex functionality of so called lexicographic stations that manipulate XML (DPS Longman [McN03]) and several companies offer dictionary writing programs of different complexity (TshwaneLex [JdS04], iLEX [Erl04] or Field Linguist's Toolbox [Too07]). However, these and similar tools are not always able to efficiently manipulate resources obtained from data-driven NLP applications. Therefore, they cannot provide a universal environment for lexical database management as well as semantic networks and ontologies. They often represent rather large systems that are quite complex which is not always an advantage. And, last but not least, some of them are rather expensive. That is why we decided to build a development framework on which the individual clients can be built – this solution is modular and flexible since the clients can be adapted for the particular purpose in a short time. The description of this development platform, called DEB II, forms the content of the third part of this chapter.

The contents of this chapter is an extension of the previous work published in [HS03, HS04, HH05a, HH05b, HPRP06, HP06a, PH06a, HPRR06, HR07, HP07, HVR08].

2.1 VerbaLex – New Comprehensive Lexicon of Verb Valencies for Czech

The beginnings of building the verb valency frame dictionary at the MU NLP Centre dates back to 1997, when Karel Pala prepared the first version of a verb valency dictionary with 15 000 entries [PŠ97]. Since then, the dictionary, denoted as Brief, has undergone a long development and has been used in various tools from semantic classification to syntactic analysis of Czech sentence [SH98]. The data in this dictionary can be entered in several mutually convertible formats:

brief format:
`jíst`[2] `<v>hTc4,hTc4-hTc6r{na}, hTc4-hTc7`

verbose format:	display format:
`jíst`[2]	
`= co`	`jíst něco`
`= co & na čem`	`jíst něco na něčem`
`= co & čím`	`jíst něco něčím`

The Brief dictionary contains about 15 000 verbs with 50 000 verb valency frames, thus making it an invaluable language resource with high coverage. However, the different verb senses are not distinguished here.

Another advance in the Czech verb valency processing came during the work on the Czech WordNet within the Balkanet project (see [Bal04]). The Czech WordNet has been supplemented with a new language resource, Czech WordNet valency frames dictionary. The new acquisition

[2]'jíst'='to eat', 'co'='what', 'na čem'='on what', 'čím'='(with) what', 'něco'='something', 'na něčem'='on something', 'něčím'='(with) something'

of this dictionary were the semantic roles and links to the Czech WordNet semantic network.

During the work on enhancing the list and adding new entries into it, we have come to the need of comparing the quality and features of the list with the valency lexicon of Czech verbs denoted as VALLEX 1.0 [SLŽ02] that was created independently of the Czech WordNet verb frames. Based on these three verb frame resources, we have designed new format of a verb frame entry named the *complex valency frame* (CVF). The resulting lexicon of Czech verbs CVFs, named VerbaLex, contains information useful for automatic computer processing of verb frames with the linguistic background.

The VerbaLex dictionary is being actively developed, checked and supplemented with new data since 2005. The coordination of the work of 4 IT developers and 15 linguists is directed by Aleš Horák with Dana Hlaváčková as the head of the linguistic team. Currently, VerbaLex contains 10 782 verb lemmata which, when gathered in synonymic groups, share 28 566 verb frames.

2.1.1 Linguistic Requirements for the VerbaLex Format

In this section, we present the substantiation of the main differences between VerbaLex and VALLEX 1.0 valency frames notation.

VerbaLex differs from VALLEX 1.0 in augmentation of the original format, detailed differentiation of valency frames and above all two-level semantic roles.[3] The headwords in VerbaLex are formed with lemmata in a synonymic relation followed by their sense numbers (standard Princeton WordNet notation, such an expression is denoted as a WordNet *literal*). The lexical units in WordNet are organized into synsets (sets of synonyms) arranged in the hierarchy of word meanings (hyper-hyponymic relations). The standard definition of synonymy says that two synonymic words can be always substituted in the context. However, the synonymy in synsets is understood like very close sense affinity of given words, the substitution rule cannot be applied in all cases here. In VALLEX 1.0, a headword is one lemma, possibly two or more lemmata in case of lemma

[3]for more details about the VerbaLex semantic roles see the Section 2.1.2

Lemma variants:

> Princeton WordNet – plan:2
> Definition: make plans for something
> VALLEX 1.0: vymyslet$_2$ / vymyslit$_2$
> VerbaLex: vymyslet:1, vymyslit:1, naplánovat:3

Word entries:

> Princeton WordNet – arrive:1, get:5, come:2
> Definition: reach a destination; arrive by movement or progress
> VALLEX 1.0: dojít$_1$
> VerbaLex: dojít:1, dorazit:1, dostat se:1, přicestovat:1, přijet:1, přijít:1

Figure 2.1: Examples of verb frame entry heads for verbs with lemma variants and for synonymic verbs.

variants.[4] Lemma variants in VerbaLex are considered as independent lemmata and they are distinguished by their WordNet sense numbers. An example of two verb frame entries in VALLEX 1.0 and VerbaLex is displayed in the Figure 2.1.

In VerbaLex, each word entry includes an information about the verb aspect (perfective – `pf.`, imperfective – `impf.` or both aspects – `biasp.`). VerbaLex valency frames are enriched with aspect differentiations for examples containing the verb used with the given valency frame. This is important in case of synonymic lemmata with different aspect:

> Princeton WordNet – wade:1
> Definition: walk (through relatively shallow water)
> VerbaLex: brodit se:2$_{\text{impf.}}$, přebrodit se:1$_{\text{pf.}}$
> frame: `AG<person:1>`$^{\text{obl}}_{\text{who\_nom}}$ `VERB` `SUBS<substance:1>`$^{\text{obl}}_{\text{(through+)what\_ins}}$
> example: přebrodil se blátem$_{\text{pf.}}$ / he wade through mud

[4]the lemmata with small phoneme alternation (in Czech) that are interchangeable in any context without any change of the meaning – `bydlet/bydlit` = to live (somewhere).

example: brodil se pískem$_{\text{impf.}}$ / he wade through sand

The constituent elements of frame entries are enriched with pronominal terms (e.g. *who*, *what*) and the morphological case number or short word.[5] This notation allows to differentiate an animate or inanimate agent position:[6]

Princeton WordNet – bump:1, knock:3
Definition: knock against with force or violence
VerbaLex: narazit:1$_{\text{pf.}}$ / narážet:1$_{\text{impf.}}$
frame: `AG<person:1>`$^{\text{obl}}_{\text{who\_nom}}$ `VERB`
`OBJ<object:1>`$^{\text{obl}}_{\text{to+what\_gen,at+what\_acc}}$
`PART<body part:1>`$^{\text{obl}}_{\text{(with+)what\_ins}}$
example: I bumped to the wall with my head
frame: `OBJ<vehicle:1>`$^{\text{obl}}_{\text{what\_nom}}$ `VERB`
`OBJ<object:1>`$^{\text{obl}}_{\text{to+what\_gen,at+what\_acc}}$
example: the car bumped to the tree

2.1.1.1 Verb Usage and Verb Classes

VerbaLex captures additional information about types of verb use and semantic verb classes. Three types of verb use are displayed in the lexicon. The primary usage of a verb is marked with abbreviation `prim`, metaphorical or figurative use with `fig` and idiomatic and phraseological use with `idiom` (this notation corresponds to the original VALLEX 1.0). The assigned semantic verb classes have been adopted from the Martha Palmer's [DKPR98] VerbNet project. The verb classes list is based on Beth Levin's [Lev93] classes with more fine-grained sets of verbs.

There are 395 classes in the current development version of VerbNet, which was provided by Martha Palmer's team. But this number seems to be too high for Czech verbs, therefore the list of verb classes is adapted to the conditions of the Czech language:

Princeton WordNet – cry:2, weep:1

[5]for the 7 grammatical cases in Czech – nom[inative], gen[itive], dat[ive], acc[usative], voc[ative], loc[ative] and ins[trumental]

[6]The prepositions in VerbaLex are stated in Czech. Here, in the presented examples, we translate them to English.

Definition: shed tears because of sadness, rage, or pain
VerbaLex: brečet:1, plakat:1, ronit:1
class: nonverbal_expression-40.2

Princeton WordNet – take care:2, mind:3
Definition: be in charge of or deal with
VerbaLex: dbát:2, starat se:2, pečovat:3
class: care-86

Princeton WordNet – be:11, live:5
Definition: have life, be alive
VerbaLex: žít:1, být:2, existovat:3
class: exist-47

2.1.2 Semantic Roles

Semantic role annotation is usually based on the appropriate inventories of labels for semantic roles (deep cases, arguments of verbs, functors, actants) describing argument predicate structure of verbs. It can be observed that different inventories are exploited in different projects (e.g. VALLEX [ŽL04], VerbNet [KDP00], FrameNet [FBS04], Salsa [BPGR06], CPA [Han04] or VerbaLex here).

The idea of semantic roles in VerbaLex has come from the specification of needs of the Czech syntactic analysis – we need a technique for distinguishing sentence constituents as a) obligatory,[7] b) typical (for the purpose of syntactic tree ranking,[8] or c) forbidden, for the sake of tree pruning.

These considerations led us to the design of the inventory of *two-level labels* which are presently exploited for annotating semantic roles in VerbaLex. VerbaLex has thus introduced a different concept of semantic roles than in the VALLEX 1.0 project.[9] The functors used in VALLEX 1.0 valency frames seem to be too general and they do not allow to distinguish different senses of verbs according to the WordNet style. We suppose that a more specific subcategorization of the semantic role tags is necessary for the needs we have defined above.

[7]with regard to the valency of another sentence constituent
[8]see the Sections 3.2.2 and 3.2.3
[9]semantic roles are denoted as functors in VALLEX 1.0

The first level of VerbaLex semantic labels contains the main semantic roles proposed on the basis of 1st and 2nd Order Entities from the EuroWordNet Top Ontology [VB^{+}98]. On the second level, we use selected particular literals (lexical units) from the set of Princeton WordNet Base Concepts with relevant sense numbers. We can thus specify groups of words (hyponyms of these literals) replenishable to valency frames. This concept allows us to specify valency frames notation with large degree of sense differentiability. The motivation for this choice is based on the fact that the Princeton WordNet has a hierarchical structure which covers about 110 000 English lexical units (synsets). It is then possible to use general labels corresponding to selected top and middle nodes and go down the hyperonymy/hyponymy (H/H) tree until the particular synset is found or matched. This allows us to see what is the semantic structure of the analyzed sentences using their respective valency frames. The nodes that we have to traverse when going down the H/H tree at the same time form a sequence of the semantic features which characterize meaning of the lexical unit fitting into a particular valency frame. These sequences can be interpreted as quite detailed selectional restrictions. Currently, we use about 40 1st-level and 200 2nd-level semantic roles in VerbaLex.

For example the literal `writing implement:1` is a hypernym for any implement that is used to write.

> Princeton WordNet – draw:6
> Definition: represent by making a drawing of, as with a
> pencil, chalk, etc. on a surface
> VerbaLex: kreslit:1, malovat:1
> frame: `AG<person:1>`$^{\text{obl}}_{\text{who\_nom}}$ `VERB`
> `ART<creation:2>`$^{\text{obl}}_{\text{what\_acc}}$
> `INS<writing implement:1>`$^{\text{obl}}_{\text{(with+)what\_ins}}$
> example: my sister draws a picture with coloured pencils,
> the famous artist was drawing his painting only with
> charcoal

The left-side valency position is most frequently occupied by the semantic role `AG`, an agent. The agent position in a valency frame is understood as a very general semantic role (functor `ACT`) in VALLEX 1.0. This label does not allow to distinguish various types of action cause. Two level semantic role labels in VerbaLex are able to define cause of action quite precisely. The main semantic role `AG` is completed by an adequate literal depending on the verb sense and valency frame. Thus, we can identify whether the agent is a person `AG<person:1>`, an animal `AG<animal:1>`, a group of people `AG<group:1>`, an institution `AG<institution:1>` or a machine `AG<machine:1>`. For some verbs with a very specific sense, hyponyms of these literals are used. For example:

Princeton WordNet – sugar:1, saccharify:1
Definition: sweeten with sugar
VerbaLex: sladit:4, osladit:1, pocukrovat:1
frame: $\texttt{AG<person:1>}^{\text{obl}}_{\text{who\_nom}}$ `VERB` $\texttt{SUBS<food:1>}^{\text{obl}}_{\text{what\_acc}}$
$\texttt{SUBS<sugar:1>}^{\text{obl}}_{\text{(with+)what\_ins}}$
example: sugar your tea with brown sugar

In VALLEX 1.0, each valency frame starts always with functor `ACT`. In our opinion, it is useful to differentiate the sense of the left-side valency position (subject position) in more detail. According to our definition of agent `AG` (somebody or something doing something actively) this position may be also occupied by other semantic roles. The subject position can contain objects `OBJ`, substances `SUBS` or a semantic role denoting abstract concepts – human activity `ACT`, knowledge `KNOW`, event `EVEN`, information `INFO`, state `STATE`. For example:

Princeton WordNet – follow:6, come after:1
Definition: come after in time, as a result
VerbaLex: přijít:25 / přicházet:25, následovat:4
frame: $\texttt{EVEN<event:1>}^{\text{obl}}_{\text{what\_nom}}$ `VERB`
$\texttt{EVEN<event:1>}^{\text{obl}}_{\text{(after+)what\_loc}}$
example: heavy rain followed flood

Princeton WordNet – fall:3
Definition: pass suddenly and passively into a state of body or mind
VerbaLex: zachvátit:2, zmocnit se:2
frame: $\texttt{STATE<state:4>}^{\text{obl}}_{\text{what\_nom}}$ `VERB`
$\texttt{PAT<person:1>}^{\text{obl}}_{\text{whom\_acc}}$
example: he fall into a depression

Table 2.1: List of 1st-level semantic roles from VerbaLex that are used in the examples.

AG	the semantic role of the animate entity that instigates or causes the happening denoted by the verb in the clause, we extended this definition for inanimate entity that does sth actively (e.g. machine)
ART	a man-made object taken as a whole
SUBS	that which has mass and occupies space
PART	a portion of a natural object, something determined in relation to something that includes it, something less than the whole of a human artifact
INS	a device that requires skill for proper use
OBJ	a tangible and visible entity; an entity that can cast a shadow
EVEN	something that happens at a given place and time
STATE	the way something is with respect to its main attributes

A large number of semantic roles inspired by EuroWordNet Top Ontology roughly correspond with the `PAT` functor in VALLEX 1.0. The `PAT` label covers quite different senses, which can be very well identified.

In our inventory, `PAT` is defined as: *the semantic role of an entity that is not the agent but is directly involved in or affected by the happening denoted by the verb in the clause* (the definition of the `patient:2` literal from the Princeton WordNet).

Princeton WordNet – experience:1, undergo:2, see:21, go through:1
Definition: go or live through
VALLEX 1.0: absolvovat$_2$
frame: $\mathrm{ACT}_1^{\mathrm{obl}}$ $\mathrm{PAT}_4^{\mathrm{obl}}$
VerbaLex: absolvovat:2, prožít:1 / prožívat:1
frame: $\texttt{AG<person:1>}^{\mathrm{obl}}_{\mathrm{who\_nom}}$ `VERB`
$\texttt{EVEN<experience:3>}^{\mathrm{obl}}_{\mathrm{what\_acc}}$
example: he underwent difficult surgery

Some second level literals cannot be adopted from the Princeton WordNet Base Concepts – especially specifications of roles considered as "classic" deep cases. These literals (e.g. `agent:6`, `patient:2`, `donor:1`, `addressee:1`, or `beneficiary:1`) do not have any hyponyms in Princeton WordNet and cannot be substituted by any word.

For such cases, the literal `person:1` is used (or another suitable literal with large number of hyponyms, e.g. `AG<person:1>`, `PAT<animal:1>`). This "classic" semantic roles are consistent with some functors in VALLEX 1.0 (`ACT`, `PAT`, `ADDR`, `BEN` etc.). A list of VerbaLex semantic roles that are used in the presented examples is displayed in the Table 2.1.

2.1.2.1 Special Semantic Roles

VerbaLex describes not only the valency and semantic frames, it also includes other relevant information about Czech verbs, such as the verb position. In a free-word order language like Czech the position of the verb within the verb frame is usually not strictly specified.

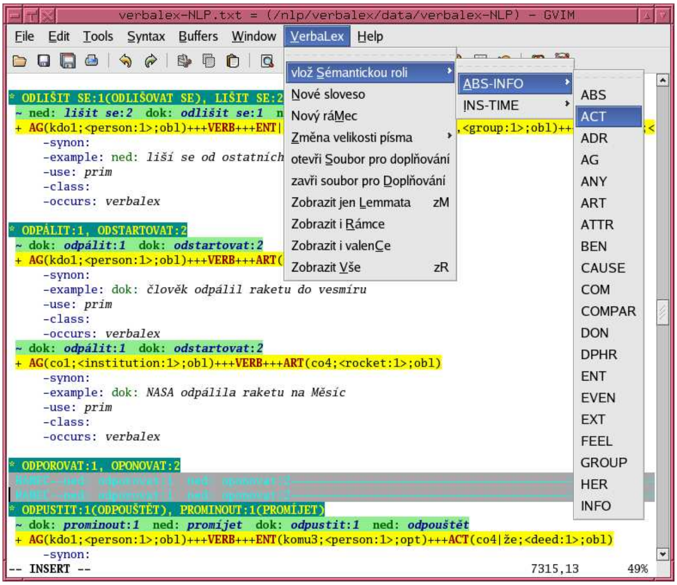
VerbaLex uses a special semantic role, `VERB`, which marks the canonic position of the verb in its verb frame. Such default verb position is not needed only for analysis of verb valencies, it can be also directly used in the process of generation of Czech sentences, e.g. as an output of a question-answering machine.

The left side of the verb position is traditionally occupied by the sentence subject, which is also marked in most of the verb frames in VerbaLex. However, there are some cases, where the verb frame has to obey different rules – e.g. sentence *Dalo se do deště* (It started to rain) cannot contain any subject. For the notation of such cases, VerbaLex uses another special semantic role `ISUB`, an inexplicit subject.

2.1.3 The Implementation of Editing and Exporting Tools

For the sake of editing and entry management in the newly adopted verb valency frame format for VerbaLex, we have implemented a new set of editing tools.

The main interactive tool for user editing of the valency dictionary, named `verbalex.sh`, is based on a highly configurable multi-platform



```
* ODLIŠIT SE:1(ODLIŠOVAT SE), LIŠIT SE:2
 ~ ned: lišit se:2  dok: odlišit se:1  n
 + AG(kdo1;<person:1>;obl)+++VERB+++ENT|                         ,<group:1>;obl)++
     -synon:
     -example: ned: liší se od ostatních
     -use: prim
     -class:
     -occurs: verbalex

* ODPÁLIT:1, ODSTARTOVAT:2
 ~ dok: odpálit:1  dok: odstartovat:2
 + AG(kdo1;<person:1>;obl)+++VERB+++ART(
     -synon:
     -example: dok: člověk odpálil raketu do vesmíru
     -use: prim
     -class:
     -occurs: verbalex
 ~ dok: odpálit:1  dok: odstartovat:2
 + AG(co1;<institution:1>;obl)+++VERB+++ART(co4;<rocket:1>;obl)
     -synon:
     -example: dok: NASA odpálila raketu na Měsíc
     -use: prim
     -class:
     -occurs: verbalex

* ODPOROVAT:1, OPONOVAT:2
 RÁMEC--ned: odporovat:1  ned: oponovat:2----------------------------
 RÁMEC--ned: odporovat:1  ned: oponovat:2----------------------------
* ODPUSTIT:1(ODPOUŠTĚT), PROMINOUT:1(PROMÍJET)
 ~ dok: prominout:1  ned: promíjet  dok: odpustit:1  ned: odpouštět
 + AG(kdo1;<person:1>;obl)+++VERB+++ENT(komu3;<person:1>;opt)+++ACT(co4|že;<deed:1>;obl)
     -synon:
-- INSERT --                                              7315,13        49%
```

Figure 2.2: The tool for editing verb valency frames dictionary in the VerbaLex format.

editor VIM [Sch07] (see the Figure 2.2). Such approach enables a linguistic expert to easily enter computer-parseable data in a fixed plain text format and still, thanks to the flexible color syntax highlighting, he or she has a full visual control of possible errors in the format.

The editing itself is not fixed to one platform, users can run the same environment under any of the current popular computer operating systems (VIM editor runs on nearly any platform).

The authoring tool `verbalex.sh` currently offers these functions to the editing user:

- free editing of the dictionary entries
- regular expression searching in the dictionary
- template-based adding of a new verb entry or a new verb frame to the current entry
- menu-based adding of a new semantic role to the current frame

- multilevel folding – hiding/displaying of valency attributes, valencies or full valency frames
- visual marking of the current frame for further inquiry
- interactive merging of definitions from two parallel sources

Moreover, the interpreted approach of the tool makes adding new features to the editing system easier to implement.

The plain text format edited by a human expert was inspired by the editing format of VALLEX 1.0 [Žab05]. This text format is in further processing transformed into an XML standard format which enables conversions into different formats used for visual checking, searching and presentation of the valency dictionary.

A good example of merging various lexical data is the work going on in the NLP Centre at FI MU where the data from Czech WordNet and Czech Valency Lexicon VerbaLex are combined together. The VerbaLex lexicon is currently being developed separately and independently of Czech WordNet using a particular XML format (see the Figure 2.3). However, the entries in VerbaLex are written in form of WordNet synsets, which enables combining the data from both these resources.

The Czech WordNet currently contains a smaller set of valency frames in plain PCDATA format (see the Figure 2.4). The current work is directed to merging the VerbaLex valency frames with the Czech WordNet synset structures.

The Czech verbs for which valency frames already exist are or will be linked to their English equivalents by means of ILI (Inter-Lingual Index). If Czech and English verbs (synsets) are linked correctly, the deep valency frames developed for Czech can be also valid for English (surface valencies are obviously different since Czech is a synthetic language whereas English is an analytic one).

The XML schema for VerbaLex also took inspiration from VALLEX 1.0, where the original schema had to be changed to suit the augmentation of the format in VerbaLex. The changes include

- adding `class` attribute to frame `slot` tag to cover WordNet basic concept literals
- including the WordNet word sense in the lemma tags
- shifting the verb aspect to `headword_lemma`, which now enumerates all the aspectual counterpart tuples. An example of such XML substructure can be found in the Figure 2.3.

```
<word_entry>
 <headword_lemmata>
   <lemma ord='1' sense='1' aspect='pf'
       aspectual_counterpart_lemma='dodávat'>dodat</lemma>
   ...
 </headword_lemmata>
 <frame_entry frame_index='1'>
   <frame_lemmata>
     <lemma  sense='8' aspect='pf'>dát</lemma>
     ...
   </frame_lemmata>
   <synonym_lemmata>
     <lemma aspect='pf' sense='1'>vložit</lemma>
     ...
   </synonym_lemmata>
   <example>dok: připojili ke smlouvě své podpisy</example>
   <use>prim</use>
   <frame_slots>
     <slot number='1' functor='AG' type='obl' class='person:1'>
       <form type='direct_case' case='kdo1' />
     </slot>
     <slot number='2' type='obl' functor='VERB'/>
     <slot number='3' functor='INFO' type='obl' class='info:1'>
       <form type='direct_case' case='co4' />
     </slot>
     <slot number='4' functor='COM' type='obl'
           class='written communication:1'>
       <form type='prepos_case' prepos_lemma='k' case='čemu3'/>
     </slot>
   </frame_slots>
 </frame_entry>
 ...
</word_entry>
```

Figure 2.3: An example of (a part of) an entry in the VerbaLex XML format for the synset `dát:8, vložit:1, vsunout:1, přidat:2, připojit:1, dodat:1` (i.e. insert:1, infix:1, enter:7, introduce:6 in the Princeton WordNet)

Synset: dát:8, vložit:1, vsunout:1, přidat:2, připojit:1, dodat:1

```
<VALENCY>
<FRAME>{dát, vložit, vsunout}
    kdo1*AG(person:1)=co4*OBJ(object:1)
       & do čeho2*OBJ(container:1)
</FRAME>
<FRAME>{dát, vsunout, přidat, vložit, dodat}
    kdo1*AG(person:1)=co4*INFO(info:1)
        & do čeho2*COM(written communication:1)
    %dodal do textu nové poznámky, přidal k článku obrázek
</FRAME>
<FRAME>{dát,přidat, připojit, dodat}
    kdo1*AG(person:1)=co4*INFO(info:1)
        & k čemu3*COM(written communication:1)
    %připojili k smlouvě své podpisy
</FRAME>
<FRAME>{přidat, připojit, dodat}
    kdo1*AG(person:1)=co4*OBJ(object:1)
        ? k čemu3*OBJ(object:1)
    %připojil hadici ke kohoutku
</FRAME>
```

Figure 2.4: An example of valency frames in the Czech Wordnet for the same synset `insert:1, infix:1, enter:7, introduce:6` as was displayed in the Figure 2.3

The resulting XML structure is then transformed into various output formats with the use of modified tools from VALLEX 1.0. The export formats are:

- HTML with navigation among the characteristic features of the dictionary entries,
- Postscript document for printing including page index of all verbs, and
- PDF, which allows navigation through the document in the same visual form as for hardcopy printing.

2.1.4 Application of VerbaLex in Syntactic Analysis

The design of VerbaLex verb valency lexicon was driven mainly by the requirement to describe the verb frame (VF) features in a computer readable form suitable for syntactic and semantic analysis. The current CVFs[10] structure contains:

- morphological and syntactic features of constituents
- two-level semantic roles
- links to Princeton WordNet and Czech WordNet hypero/hyponymic hierarchy
- differentiation of animate/inanimate constituents
- default verb position
- verb frames linked to verb senses
- VerbNet classes of verbs.

We are currently testing the application of VerbaLex in the syntactic analyzer `synt` (see the Chapter 3) that is designed for parsing real-text sentences. For the detailed description of the verb frame extraction process see the Section 3.4.1.

The system processing can be presented on an example sentence – see the syntactic tree in the Figure 2.5 and the textual output of the part of the system that implements the VFE algorithm in the Figure 2.6. The system first identifies the verb rule constituents (`nterms`). Then the corresponding groups, i.e. the actual sentence constituents that will play the role as verb frame arguments, are extracted from the forest of values. Groups usually do not correspond to nterms one-to-one, since they are stored within non-terminals deeper in the forest and not directly in the verb rule. This part of the VFE algorithm has unfortunately exponential time complexity, however, for common sentences the depth of the verb frame constituents is not more than three levels, so the actual running times are usually within fractions of seconds. After the identification of the groups, the algorithm looks for possible subjects – this is not as easy as it may look at the first sight, since the sentence subject can be expressed not only by a noun phrase in nominative (which is the most frequent option in Czech), but also by e.g. prepositional phrase or verb infinitive. If no possible subject is found, the algorithm supplies a pronoun for an inexplicit subject with the gender corresponding to the verb. The Clause valency list displays all possible combinations of the

[10] Complex Valency Frames

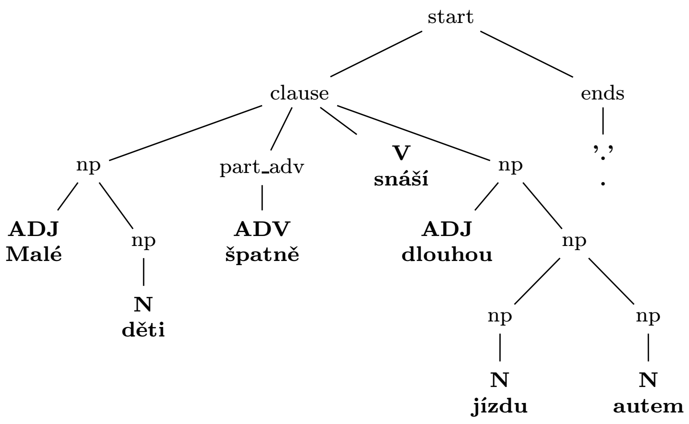


Figure 2.5: Syntactic tree of an example input sentence “`Malé děti špatně snáší dlouhou jízdu autem.`” (Small children badly withstand long journey by car.)

```
verb_rule_schema:  3 nterms, '#2'
   nterm 1:  k1gNnPc1
   nterm 2:  k5eAp3nPtPmIaI
   nterm 3:  k1gFnSc4
   group 1:  0,2, npnl -> .{ left_modif } np .  k1gMnSc1
                   "malé děti"
   group 2:  2,3, ADV -> .'špatně' .  k6xMeAd1
   group 3:  4,7, npnl -> .{ left_modif } np .  k1gFnSc4
                   "dlouhou jízdu autem"
possible subjects:  #1
Clause valency list:
     snášet <v>-#2:(1)hH#1:(0)hPTc1-#3:(2)hPTc4
     snášet(0) <v>#1:(1)hH-#2:(2)hPTc4
Verb valency list:
     snášet <v>#2:hH-#1:hPTc4
     snášet <v>#1:hPTc4
Matched valency list:
     snášet(0) <v>#2:(1)hH-#1:(2)hPTc4
```

Figure 2.6: The output of the verb frame extraction algorithm during the example sentence analysis.

translations of the verb arguments found into verb frame patterns. This list is then intersected with the list of lexicon entries for the verb to obtain the Matched valency list as a result of the VFE algorithm.

The effectiveness of the syntactic analysis with the VFE algorithm was measured on approximately 4 000 Czech corpus sentences with the median of 15 words per sentence and the Clause valency list contained 11 possible verb frames with the running time of 0.07 seconds per sentence.

2.2 VisDic – Off-line WordNet Editor

As the developers of Czech WordNet within EuroWordNet project we came to the conclusion that a new tool for WordNet browsing and editing has to be developed rather quickly. At the same time we realized that it was necessary to look for the solution that would also support establishing the necessary standards for WordNet like lexical (knowledge) databases. Thus we decided to develop a new tool for WordNets based on XML data format, which can be used for lexical databases of various sorts. The tool is called VisDic and it has been implemented in 1999–2004 in the Natural Language Processing Laboratory at the Faculty of Informatics, Masaryk University for both Windows and Linux platform [PP02, HS03].

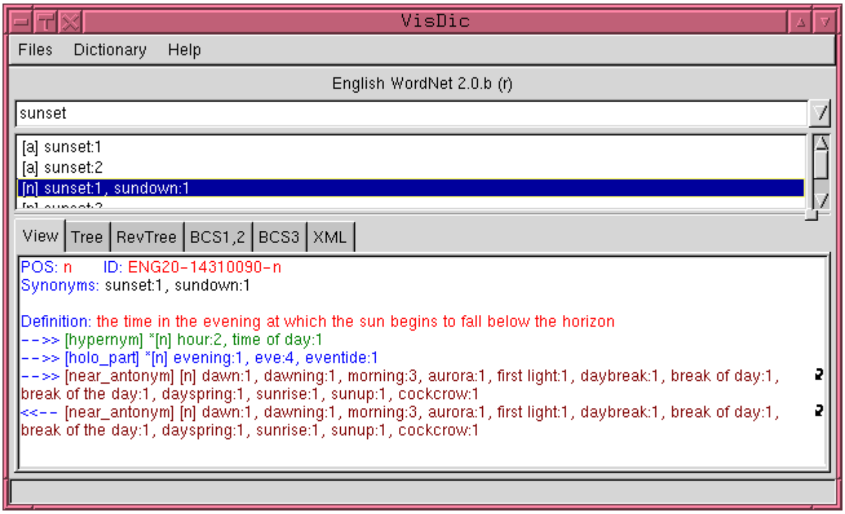
2.2.1 Basic Functionality

VisDic was developed as a tool for presentation and editing (primarily WordNet-like) dictionary databases stored in XML format. Most of the program behaviour and the dictionary design can be configured. With these capabilities, we can adopt VisDic to various dictionary types—monolingual, translational, thesaurus or generally linked WordNet lexicons.

2.2.1.1 Multiple Views of Multiple WordNets

The main working window is divided into several dictionary panels. Each panel represents a place for entering queries and browsing context of one specified WordNet dictionary. The panels can display different WordNets as well as multiple contexts of the same dictionary.

The contents of a panel offers, besides the query input and matching results list, a set of overlapping notebook tabs each of which represents



```
[a] sunset:1
[a] sunset:2
[n] sunset:1, sundown:1
```

```
POS: n    ID: ENG20-14310090-n
Synonyms: sunset:1, sundown:1

Definition: the time in the evening at which the sun begins to fall below the horizon
-->> [hypernym] *[n] hour:2, time of day:1
-->> [holo_part] *[n] evening:1, eve:4, eventide:1
-->> [near_antonym] [n] dawn:1, dawning:1, morning:3, aurora:1, first light:1, daybreak:1, break of day:1,
break of the day:1, dayspring:1, sunrise:1, sunup:1, cockcrow:1
<<-- [near_antonym] [n] dawn:1, dawning:1, morning:3, aurora:1, first light:1, daybreak:1, break of day:1,
break of the day:1, dayspring:1, sunrise:1, sunup:1, cockcrow:1
```

Figure 2.7: An example of freely defined text view of a WordNet entry

one kind of view of the same entry from the list of results. The order, the type and even the content of each notebook tab is specified by the user in the configuration files (see the Section 2.2.3). The main types of views are described in the following sections.

2.2.1.2 Freely Defined Text Views

The content of the Text View notebook tab is entirely built from the user definition that follows the XML structure of the WordNet entry. The editor can thus present an easily readable view of the entry with highlighting important parts of the entry content (see the Figure 2.7).

2.2.1.3 Edit

The editing capabilities allow to give the user a full control over the content and linking of each entry in the WordNet hierarchy. To prevent a user from moving the entry as an object in the multicolored spider web of the linkage relations, the linguist rather specifies all the links in a textual dialog, where all the bindings are displayed in one place with consistency checks after each change request.

The actual contents of the Edit notebook tab is also entirely driven by user instructions in the configuration, where each editing field is given a textual label and is assigned to an XML tag from the entry structure.

2.2.1.4 Tree and RevTree

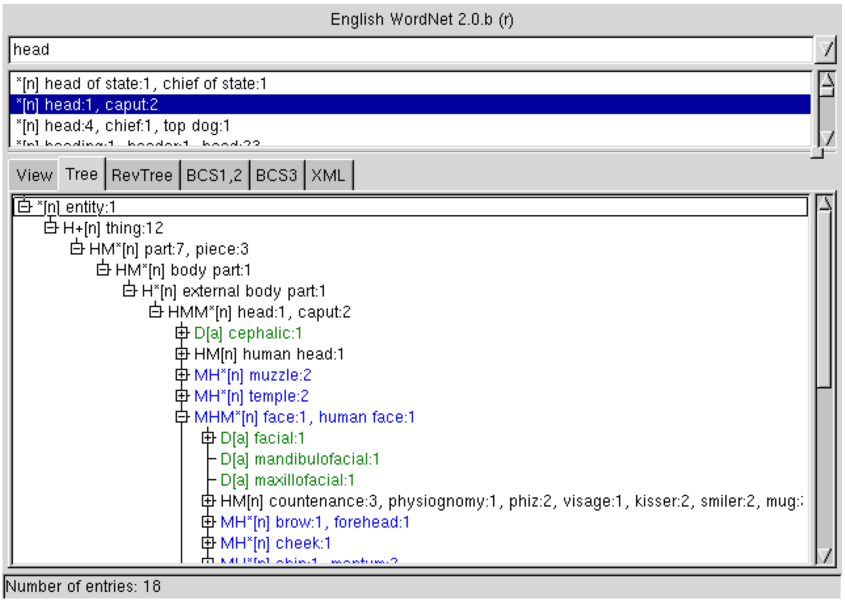
WordNet dictionaries are specific with a heavy network of various kinds of relations between the dictionary entries with the function to capture the ontology relations of the underlying natural language.

Navigation in such environment is thus a crucial point of a successful linguistic work with WordNet data. Since the linkage relations generally do not need to obey any rules, that could make the resulting structure to be an arbitrary directed graph. VisDic implements a browsing mechanism for general graphs. The navigation process works with two interconnected notebook tabs, which always both start at the same dictionary entry and display its position in the graph represented as a breadth-first path trees of all the linkage relations that lead from the entry to other entries in the dictionary. Each of the notebook tabs displays mutually opposite linkage relations, allowing the user to choose the direction of graph navigation in every step.

To facilitate the orientation and to help to position the entry in the WordNet hierarchy, the navigation also displays the path from the entry to its top in the hyper-hyponymic relation tree (see the Figure 2.8). For more advanced navigation the linguist may also use advanced tree browsing techniques (described in the Section 2.2.2.3).

2.2.1.5 Query Result and External File Lists

Common actions in WordNet creation and editing often include processing of a subset of entries based on certain criteria. VisDic offers a suitable kind of views for this situation, which allow to prepare a notebook tab with a list of entries matching any user specified query or a list of entries identified by entry-IDs gathered in a plain text file.



```
*[n] head of state:1, chief of state:1
*[n] head:1, caput:2
*[n] head:4, chief:1, top dog:1

*[n] entity:1
  H+[n] thing:12
    HM*[n] part:7, piece:3
      HM*[n] body part:1
        H*[n] external body part:1
          HMM*[n] head:1, caput:2
            D[a] cephalic:1
            HM[n] human head:1
            MH*[n] muzzle:2
            MH*[n] temple:2
            MHM*[n] face:1, human face:1
              D[a] facial:1
              D[a] mandibulofacial:1
              D[a] maxillofacial:1
              HM[n] countenance:3, physiognomy:1, phiz:2, visage:1, kisser:2, smiler:2, mug:
              MH*[n] brow:1, forehead:1
              MH*[n] cheek:1
Number of entries: 18
```

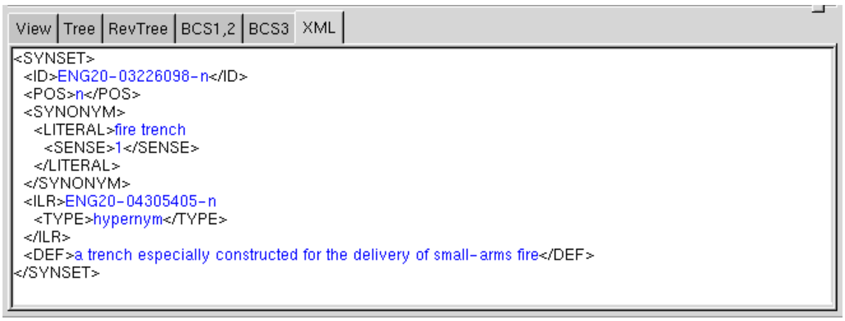
Figure 2.8: The tree-like navigation in the WordNet linkage relations graph

2.2.1.6 Plain XML View

Sometimes users need a thorough view into the data structures contained in the dictionary entry. XML View notebook tab offers this possibility. In this view, the user can see a graphically structured XML text, which represents the entry structure as it is stored in the dictionary (see the Figure 2.9).

2.2.2 Advanced Functionality

The basic functionality described in the previous section generally conforms to any XML based dictionary. However, linguistic work specialized to WordNet creation and editing requires some more specific and more sophisticated functions in the editor.



```
<SYNSET>
  <ID>ENG20-03226098-n</ID>
  <POS>n</POS>
  <SYNONYM>
    <LITERAL>fire trench
      <SENSE>1</SENSE>
    </LITERAL>
  </SYNONYM>
  <ILR>ENG20-04305405-n
    <TYPE>hypernym</TYPE>
  </ILR>
  <DEF>a trench especially constructed for the delivery of small-arms fire</DEF>
</SYNSET>
```

Figure 2.9: Raw XML view of a synset entry.

2.2.2.1 Synchronization

Within the creation of a national (e.g. Czech) WordNet, which would correspond to the English WordNet as a primary reference, one of the most frequent operation is a lookup of a dictionary entry (synset) from one WordNet in another dictionary. Such lookup uses either the `SYNSET.ID` tag (as a direct equivalent of the entity) or one of the, so called, equivalence tags (or attributes) defined in the configuration. An example of such tag may be `REVMAP` or `MAPHINT` used to help the linguist to process ambiguous link references between various versions of English WordNet.

The lookup function in VisDic can work in two modes: as an instant (one time) lookup — the *Show (by)* operation, and also as a firmly established link between two notebook tabs called the *AutoLookUp (by)*. In case of AutoLookUp, any move to another dictionary entry in the source notebook tab leads to an automatic lookup of the new entry in the destination dictionary. VisDic allows to have any acceptable combination of AutoLookUps among all the notebook tabs.

2.2.2.2 Editing Support

The efforts of unifying national WordNets based on the English WordNet in many cases lead to copying of synset information between different language dictionaries. Such functionality in VisDic is split into two common situation — either the `SYNSET.ID` of an existing synset is to be unified with the ID of the English synset (*Take key from* operation) or a whole new entry is to be copied to another dictionary (*Copy entry to*).

2.2.2.3 Tree Browsing

The basic navigation in related synsets (in some cases reduced to the hypernymic and hyponymic relations tree) is supplemented with two important WordNet operations — *Topmost entries* and *Full expansion.*

The Topmost entries operation identifies all synsets, which are (in the tree subset of linkage relations) found as the roots of relational hierarchy, i.e. are not hung below any other synset. This helps the linguist to identify the level 1 entries as well as so far unfiled entries.

The Full expansion allows the user to see all possible descendants of a selected synset in the linkage relations graph. During the operation cycle detection techniques check any violations of tree properties in the graph. Some relations can be also configured to be left out from the full expansion process.[11]

2.2.2.4 Consistency Checks

Semi-automatic processing, which often takes part in the national WordNets creation, as well as common human processing of the data inevitably brings in the possibility of mistakes. The inconsistencies, which may be revealed as duplicates, are controlled by VisDic consistency checks, which contain

- check duplicate IDs
- check duplicate literals and senses
- check duplicate synset literals
- check duplicate synset links

These checks allow the linguist to identify the most common errors e.g. after merging data from various sources.

2.2.2.5 Journaling

The work on a large and representative national WordNet usually employs more than one linguist working on the data. The synchronization of

[11]see also *Visual Definitions* in the Section 2.2.3.2

the resulting dictionary is made possible in VisDic with the usage of *journaling.*

During the work with VisDic, any change of the data is marked in a journal file. Each journal file is specific to one dictionary and one user at a time. Such journal file can then be "applied" to the dictionary data and merged with the original. In this way, the simultaneous work of several linguists can be easily interchanged with a common data source.

2.2.3 XML Configuration

Most of the functionality in the VisDic WordNet editor can be adapted to local needs by means of its configuration files. All settings for the VisDic application are stored in several XML files.

2.2.3.1 Global Configuration

The main configuration file (`visdic.cfg`) serves for global application data storage such as the list of dictionaries, the list of views, fonts, colors or query history. All information is stored in an XML structure.

The first-level subsections of the global configuration are:

colors In the `COLOR` section the user can define colors which are then referenced by its name in dictionary configuration files. Each color is enclosed in its name tag and consists of three hexadecimal values separated by commas, representing consequently its red, green and blue components. Each value can be in range $\langle 0x0000, 0xffff\rangle$.

fonts The `FONT` section defines fonts which can be referenced by the defined names from dictionary configurations. Each font definition is enclosed in its name tag and its value correspond to the font string description.

application settings The `APPL` section contains all global data that are related to the application state. The most common settings that can be found here are:

- `DICT` – path to a dictionary that is presented in the list offered to the user.
- `OPEN` – ordinal number of a dictionary that should be opened in one notebook tab.

- `AUTOLOOKUP` – definition of a synchronization link between two notebook tabs.
- `SIZE` – size of the notebook tab in percentage of the main window width.
- `HIST` – history of last queries that were entered by the user in a specific notebook entry line.

A shortened example of a global configuration file is displayed in Figure 2.10.

2.2.3.2 Dictionary Specific Configuration

Each WordNet dictionary has its special configuration file (named *dictionary*`.cfg`), which enables the linguist to set up most of the texts displayed in the application as well as the content of notebook tabs specific to the particular dictionary with respect to the XML structure of the entries.

The configuration contains attribute settings of the dictionary and sections describing the layout of the dictionary views. The main attributes in the dictionary configuration are:

- `NAME` – full name of the dictionary. This name is presented to the user in various places in the application, e.g. on the top of the dictionary notebook tab.
- `SHORT_NAME` – short name of the dictionary.
- `MAIN_TAG` – the default XML tag in the user queries (e.g. `SYNSET.SYNONYM.LITERAL`).
- `MAX_QUERY` – limit of the number of results of a query.
- `MAX_VIEW` – limit of the number of characters displayed in the user defined text view.
- `CHARSET` – name of a character set indicating the encoding of the dictionary. This information is necessary for correct manipulation with the dictionary in some systems.

The rest of the dictionary configuration file contains sections defining the list of the available dictionary views and their content or the list of duplicate checking actions in the application menu.

```
<?xml version="1.0"?>
<CONFIG>Visdic general configuration file
  <COLOR>Colors definition
    <BLACK>0x0000, 0x0000, 0x0000</BLACK>
    <WHITE>0xffff, 0xffff, 0xffff</WHITE>
    <RED>0xffff, 0x0000, 0x0000</RED>
    ...
  </COLOR>
  <FONT>Fonts definition</FONT>
  <APPL>
    <DICT>/nlp/wn/visdic/data/eng20/wneng20</DICT>
    <DICT>/nlp/wn/visdic/data/cze/wncze</DICT>
    ...
    <AUTOLOOKUP>v2-v1</AUTOLOOKUP>
    <OPEN>1
      <SIZE>43</SIZE>
      <HIST>
        <LINE>trench</LINE>
        <LINE>house</LINE>
        ...
      </HIST>
    </OPEN>
    <OPEN>5
      <SIZE>57</SIZE>
      <HIST>
        <LINE>pes</LINE>
      </HIST>
    </OPEN>
  </APPL>
</CONFIG>
```

Figure 2.10: The global configuration example (. . . stands for shortened parts).

Visual Definitions The `VISUAL` section describes the way, how to display dictionary entries. Definitions are enclosed in tags corresponding to their names. VisDic uses primarily two special visual definitions. The first is called `VISDIC_SHORT` and it presents the entry in a short one-line format (e.g. list of all entries matching the query or within a tree view). The second visual definition, named `VISDIC`, describes the content of the user defined text view, i.e. it presents the entry in a more descriptive way.

Each tag from the entity XML structure can be displayed in its own way. The definition contains C-like string format specifications consisting of a string in double quotation marks and other parameters. These parameters have the following meaning:

- `%c` – color name (taken from `visdic.cfg`), changes the current color.
- `%f` – font name (taken from `visdic.cfg`), changes the current font.
- `%s` – string, it can be `@tag:name` for the current tag name, or `@tag:value` for the tag value.
- `%i` – includes the output of processing of subtags.
- `%K+` – in the tree view, stop expanding the *Full expansion* view under the current entry.
- `%K` – in the tree view, delete the current line from the tree.

The format string can include parts that are displayed only under a certain condition. The available conditions are

- `\\{^...\\}` – display ... only if the tag is the first in the list.
- `\\{$...\\}` – display ... only if the tag is the last in the list.
- `\\{*...\\}` – display ... only if the tag is not the last in the list.

An example of usage of the conditional parts of the format string can be a comma separated list of literals with their senses:

```
<SYNONYM>"%i"
    <LITERAL>"%s:%i\\{*, \\}",@tag:value
        <SENSE>"%s",@tag:value</SENSE>
    </LITERAL>
</SYNONYM>
```

The visual definition of each XML tag can contain a test for the value of the tag in the form `<TAG>="value":"result"</TAG>`. For instance, various type of WordNet relations between synsets can be transcribed in colored one-letter acronyms like this

```
<ILR>"%i"
      <TYPE>="hypernym":"%cH",BLACK</TYPE>
      <TYPE>="holo_member":"%cM",BLUE</TYPE>
      <TYPE>="derived":"%cD",DARK_GREEN</TYPE>
      <TYPE>"%c[%s]",RED,@tag:value</TYPE>
</ILR>
```

A special tag named `DEFAULT` stands for any tag. It is used for tags that do not have their own definitions.

Views The `VIEW` section specifies the design of notebook tabs. Each tab is described by one `LIST` subsection. Each tab has its own name in the `NAME` tag and its own type in the `TYPE` tag. According to the type, the `LIST` subsection can include other specifications of the tab content:

- `XML` view has no other options. It just displays the XML structure of a dictionary entry.
- `USER` type has `DEF` tag referencing the name of a visual definition of the user defined text view.
- `TREE` type contains two tags specifying parent and child link tags in the dictionary and the `DEF` tag for the visual definition used in the presented tree-like ordering of entries.
- `EDIT` type describes the form fields for editing one dictionary entry. The subsection contains `ITEM` or `BUTTON` tags. Items refer to XML tags in `TAG`, each has its own head label in `HEAD` and its own item type in `TYPE`. The appearance of the form field is specified in the `EDIT` tag. It can be a single line entry (`ENTRY`), a multi line entry (`TEXT`) or a checkbox (`CHECKBOX`). All form fields used for editing the link or reverse link tags[12] will be displayed as combo boxes with an arrow >> on the right side of the box, which allows the user to

[12]see the field type `R` in the Section 2.2.3.3

navigate to the referred entry. All form fields that represent a tag which can occur more than once are supplemented by two buttons + and -. These buttons are used for adding another instance or removing the current instance of the tag.

The `BUTTON` tags define buttons that run one of the storage actions. Each button has its label specified in the `TEXT` tag and its type in the `TYPE` tag. The type can be either `NEW` for creating the new entry, `DELETE` for deleting the current entry or `UPDATE` for saving the edited entry content.

- `WORD` type view presents a list of all words from the dictionary that can be found among values of the given tag.
- `ENTR` type view is a list of entries that meet a condition given by the user query in the `QUERY` tag.

Main Menu Actions The `MENU` section describes a list of dictionary-specific actions which can be added to the VisDic main menu. All these actions will be appended to the *Dictionary* submenu.

An example of the actions that can be specified in the `MENU` are the duplicate checking actions. These actions are looking for duplicate values within the dictionary, either among entries or within a single entry. The action definition is enclosed in the `DUPL` tag. The `TYPE` tag chooses the kind of comparison – `ENTR` for comparing entries or `ITEM` for comparing items within the range of one entry. The `NAME` contains a name of the action, which will be displayed in the menu. The `TAGS` tag enlists all tags that are included in the duplicate checking, more tags are separated with the vertical bar sign '`|`'. If a tag begins with a dot '`.`', then the tag is considered as a subtag of the previous tag.

Examples of the duplicate checking actions are:

- searching for all entries (synsets in WordNet) having the same `SYNSET.ILI` value

```
<DUPL>
    <TYPE>ENTR</TYPE>
    <NAME>Check duplicate ILI numbers</NAME>
    <TAGS>SYNSET.ILI</TAGS>
</DUPL>
```

- identification of all pairs *literal*:*sense* in WordNet stored in more than one synset. Here, the `.SENSE` tag corresponds to `SYNSET.SYNONYM.LITERAL.SENSE` subtag of the `SYNSET.SYNONYM.LITERAL` tag

```
<DUPL>
    <TYPE>ENTR</TYPE>
    <NAME>Check duplicate literals & senses</NAME>
    <TAGS>SYNSET.SYNONYM.LITERAL|.SENSE</TAGS>
</DUPL>
```

- finding all literals in WordNet that occur more than once in one entry (synset)

```
<DUPL>
    <TYPE>ITEM</TYPE>
    <NAME>Check duplicate synset literals</NAME>
    <TAGS>SYNSET.SYNONYM.LITERAL</TAGS>
</DUPL>
```

2.2.3.3 Dictionary Definition

Each dictionary has, besides its configuration file, an associated definition file named *dictionary*`.def`. This file describes the XML structure of the dictionary. The structure of the definition file contains features that are specific to the WordNet-like XML dictionaries.

The definition file format is a plain text with each row corresponding to one XML tag. The line format is

level tag min max type args

where the corresponding fields contain

- *level* – the tag level (0 for the top level).
- *tag* – the tag name.
- *min* – minimal number of occurrences of the tag within its supertag.
- *max* – maximum number of occurrences of the tag within its supertag (-1 means infinite number).

```
0 SYNSET              1      1     N
  1 ID                1      1     K
  1 POS               1      1     N
  1 SYNONYM           1      1     N
    2 LITERAL         1     -1     N
      3 SENSE         1      1     I @maxbyparent+1
      3 LNOTE         0      1     N
  1 ILR               0     -1     L
    2 TYPE            1      1     N
  1 RILR              0     -1     R SYNSET.ILR
  1 BCS               0      1     N
  1 DEF               0      1     N
  1 USAGE             0     -1     N
  1 SNOTE             0     -1     N
  1 STAMP             0      1     N
```

Figure 2.11: An example of a dictionary definition file.

- *type* – the kind of the tag. It can be one of
  - `N` – normal text entry.
  - `I` – integer number entry. In the *args* column a function for the default value can be stated.
  - `K` – key value uniquely identifying the entry. Such key is used by the following `L`, `R`, and `E` kinds of tags.
  - `L` – link to another synset, it represents a semantic relation.
  - `R` – similar to `L`. It is defined as reversed link specified in the *args* column.
  - `E` – contains an external information stored in another dictionary. The name of the external tag and the path to the dictionary are contained in the *args* columns. The path is absolute or relative to the VisDic initial directory, not relative to the dictionary path.
- *args* – extra arguments for some kinds of tags.

An example of a dictionary definition file can be found in the Figure 2.11.

2.3 DEBVisDic and other DEB Platform Applications

VisDic, during its rather short history, has already proved its suitability for lexical database creation. The main power of VisDic manifests itself especially in development of highly interlinked databases such as WordNet. Its unique features have assured VisDic the leading role in many WordNet editing projects.

In comparison with previous WordNet tools, VisDic exploits XML data format thus making the WordNet-like databases more standard and exchangeable. Not only that, thanks to the XML data format used and to its dictionary specific configurability VisDic can serve for developing various types of dictionaries, i.e. monolingual, translational, thesauri and multilingually linked WordNet-like databases. The experience with the VisDic tool during Balkanet project has been positive [HS04] and it was used as the main tool with which all Balkanet WordNets were developed.

VisDic, however, has its disadvantages, particularly it is not based on the client/server architecture and it does not allow to associate various attributes with literals and handle the links between them. It can work with links only between synsets which is a limiting feature for enriching WordNets with various sorts of information, e.g. in Czech with word derivation relations existing within one part of speech as well as across them.

The experience with VisDic has led us to more systematic research into the usage of XML data formats within the field of the computational lexicography. In parallel, we also pay attention to the relations between WordNets and Semantic Web. This interest gives us a strong motivation for studying the properties of the XML data formats and tools for working with them.

Thus we set as our task to design and implement a more universal dictionary writing system that could be exploited in various lexicographic applications to build large lexical databases. The system has been called Dictionary Editor and Browser (further DEB) and in its current version (named DEB II) is used in several larger lexicographic projects that are described further. The design of DEB allows us to use it advantageously also for building WordNet-like databases.

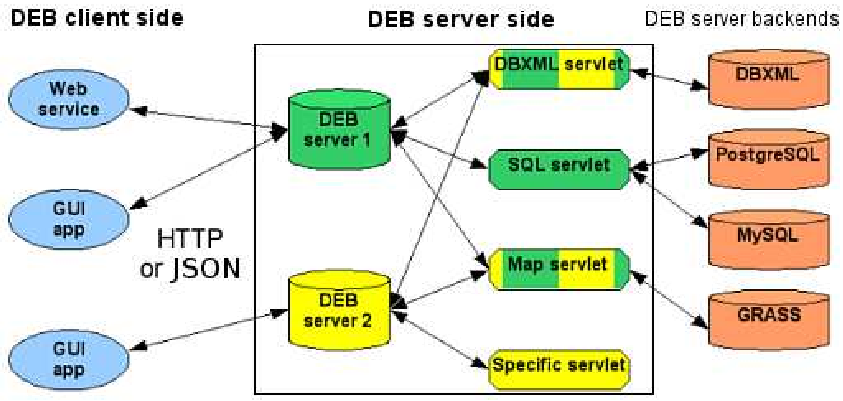


Figure 2.12: The schema of the DEB II platform architecture

2.3.1 The Features of the Platform for Lexicographers' Tools

The acronym DEB II denotes a platform or framework for building (especially) dictionary writing applications. It is based on client/server architecture, thus the application falls into two parts (see the schema on the Figure 2.12). The server includes majority of the required functions, each client part on the other hand serves as a user graphical interface which transfers user's requirements to the server that returns the demanded data. The server part is built from small parts, called *servlets*, which allow a modular composition of all services.

The clients communicate with servlets using HTTP requests in a manner similar to recently popular concept in web development called AJAX (Asynchronous JavaScript and XML [RM98]). The data are transported (using plain HTTP) in the RDF, generic XML or plain-text formats or they are marshalled using JSON (JavaScript Object Notation [Cro06]) data structure encapsulation.

The actual data storage backend on the server side is provided by Oracle Berkeley DB XML [DBX07, CRZ03, BS05], which is a native XML database providing XPath and XQuery access into a set of document containers. The metadata are stored in widely-used Berkeley DB embedded database which runs on many systems and devices ranging from Linux and Windows operating systems to mobile phones. Oracle Berkeley DB XML comes in form of a C++ library with interfaces to many scripting languages.

Since the client applications are mostly oriented to the graphical user interfaces (GUI), we have decided to adopt the concepts of the Mozilla Development Platform [O+02]. Firefox Web browser is one of the many applications created using this platform. Other applications include Mozilla Thunderbird mail client, Netscape Web browser, Komodo integrated development environment or Nvu web page editor.

The Mozilla Cross Platform Engine provides a clear separation between application logic and definition, presentation and language-specific texts. The application design is simple and allows the possibility of concurrent work of different team members which leads to significant time savings.

Mozilla platform is open source free software which ensures that it will stay free and its development will continue. Every new major version adds more features and possibilities. Also, thanks to open source design, there is a large number of free extensions of existing applications or the platform itself. Mozilla developers pay much attention to security and any reported bugs are promptly fixed.

Applications built on the Mozilla platform are working within many operating systems, actually any OS on which Mozilla runs (i.e. officially Windows, Linux, and Mac OS X, unofficially many others).

The platform also provides easy way (both for developers and users) for application installation and update.

The main "programming language" used for the GUI design of the DEB clients is called XUL (XML User-interface Language, pronounced "zool"). XUL is a user interface description language based on XML. It allows relatively simple creation of cross platform applications with possibility of easy customization of design, texts and localization. XUL itself is aimed mostly on creation of user interfaces, e.g. windows, buttons or toolbars, but it incorporates wide range of standardized technologies:

- Cascading Style Sheets (CSS) for describing the graphic appearance of the application,
- JavaScript as a programming language for simple application logic,
- Document Object Model (DOM), XSLT and XPath to work with HTML and XML documents,

- DTD for easy localization,
- RDF as data source.

2.3.1.1 The DEB Server Side

The server side of DEB is implemented in the programming language called Ruby [TH01, RUB07]. Ruby (originating in Japan) is an object-oriented, interpreted programming language with weak type checking.

The DEB server uses also various additional libraries, both pure Ruby and interfaces to C/C++ libraries:

- REXML (XML processing) and the WEBRick HTTP and SOAP4R SOAP servers (client–server communication). These modules are pure Ruby.
- Oracle Berkeley DB XML API (storage backend).
- ICU (International Component for Unicode) by IBM – language dependent character manipulation, sorting and formating [ICU07], libxslt and libxml2 (XSLT and additional XML processing) from the GNOME project. We actively participate in the development of both ICU and libxml2/libxslt bindings to the Ruby programming language.
- the GRASS geographical information system[13] (GIS) interface used in several client applications for displaying the geographical linkage of the linguistic data.
- SQL interface (connection to classical relational databases, used e.g. for the database of geographical data).

The DEB server suite runs on the Linux operating system, currently it is tested with Ubuntu Dapper on Intel x86 and AMD64 architectures, but it should generally run on any recent UNIX-based system (including Mac OS X).

Current DEB server modules (i.e. the servlets) include:

- generic document servlet – serves data from a DB XML container, supports querying the database, fetching individual documents and storage of documents or XSLT transformation of the output;

[13]see [NM04, GRA07]

- SQL servlet – provides interface to relational data in PostgreSQL (or other SQL) database;
- various specific servlets based on generic document servlet – provide additional function over XML data stored in the DB XML container;
- GRASS servlet – provides interface to the GRASS GIS, it is used for map generation;

2.3.2 Assets of the DEB Platform

The DEB platform is based on client-server architecture, which brings along a lot of benefits. All the data are stored on the server and a considerable part of the functionality is also implemented on the server, while the client applications can be very lightweight.

This approach provides adequate basis for team cooperation; data modifications are immediately seen by all the users. The server part also provides authentication and authorization mechanisms.

The server can offer different interfaces using the same data structure and these interfaces can be reused by many client applications. For example, several client applications use the same interface to query XML based dictionaries (with different underlying structure).

Although the clients are usually created using the Mozilla platform, the client software can be implemented in any way – it may be coded in any programming language or may even look only as a simple web page.

One of the main benefits of developing a dictionary writing system on the DEB platform is the homogeneity of the data structure and presentation. If the application administrator commits a change in the data presentation, this change will automatically appear in each client software. And of course, any data flaws discovered can be instantly corrected, there is no need to change the client software or provide new data files to each client.

The data sources can be implemented with different structures, that the server transforms seamlessly to a homogeneous form, which is then provided to client applications.

Of course, a drawback of the client-server architecture is that an operating server is necessary for a fully functional application. However, in special situations, the server can be installed within a local environment, or for the possibility of simple off-line WordNet editing, the client may work in a degraded manner without the instant connection to the server.

2.3.3 The DEB Administration Interface

Initially, the DEB server was developed with just command-line management of dictionaries and administration of user passwords for authentication. The configuration was realized by structured text files and data processing scripted programs.

After the client applications have spread to more users world-wide and have been used, e.g., in several national WordNet projects (Dutch, Polish, Hungarian, Slovenian or Afrikaans WordNets), a more sophisticated administration interface for the DEB users and dictionaries was created by Adam Rambousek in the MU NLP Centre under the supervision of Aleš Horák. The interface was gradually transformed into a general and complex dictionary management application for the whole DEB server.

2.3.3.1 Overall Design Goals

The DEB server packages are currently being deployed on several servers in different organizations and often more than one user need to administer a single DEB server without having a direct server access. Thus, the administration interface must be accessible remotely and without any special tools. The best choice for this task is a web-based interface, where the user needs just a web browser.

The interface should support easy administration of all the server areas. Of course, the main area of a dictionary management server is the dictionary management. Each dictionary is described with several basic attributes, like its name and code, the filename of its storage in the DB XML database, its dictionary type, the XML schema or indexed elements or XSLT templates for output displaying. Also, some projects may need extra specific settings – e.g. the DEBVisDic clients need to store information about the inter-dictionary links. After the dictionary is set up, the interface has to support import and export of XML data into and from the DB XML format.

When the administrator sets up the server dictionaries, these can be grouped to "services." A service is one individual part of the DEB server, usually used for one particular project. For example, DEBVisDic or

DEBDict are separate services, but they share the same base libraries and management database. Several services can access the same dictionaries, each providing different view on the data.

The user accounts are shared between all the services. Thanks to the database sharing between services, each user needs just one account for all the services he or she may use. The administrator can restrict access to selected services and for each service, more detailed access permissions can be set for each dictionary (read-only, read-write, update, . . . , see the Figures 2.13 and 2.14). The actual usage of the dictionary access permissions depends completely on the service implementation. This means, one service can ignore permissions at all and another service can use complex access rights.

Apart from access rights, the user account management provides all the needed functions – it allows to create, modify and delete user accounts. Each user can log-in to the administration interface and change his or her password. In case the user forgets a password, he or she can ask for a new random password.

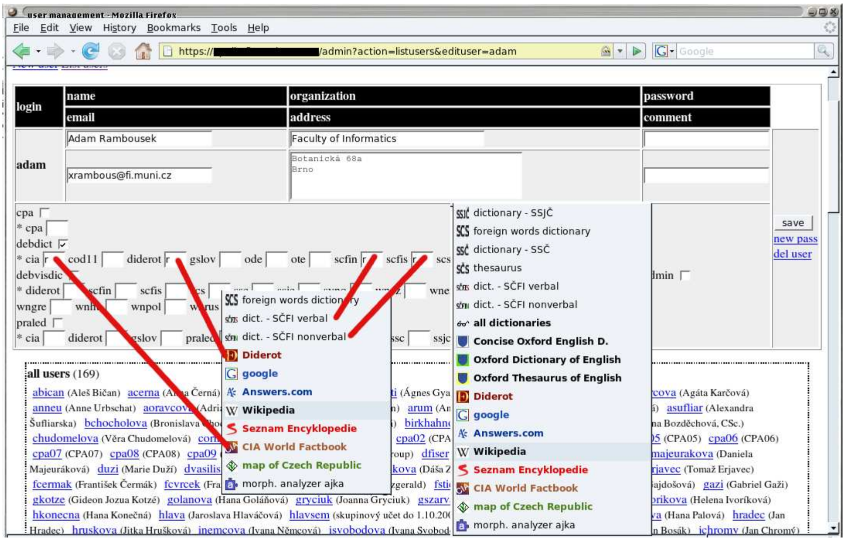
To ease the deployment of the DEB platform, we are experimenting with automated creation of the client applications. Now, the server is able to create straightforward applications based on the Relax NG Schema [vdV03] of the dictionary, and we are aiming at automated creation of client packages for new national WordNets.

Another very useful feature is uploading the client source files onto the server using the web interface. This way, the administrator can easily modify web page templates (XSLT) or other files without the need of direct (FTP, SSH) access to the server.

The server administration interface is based on the same postulates as the other DEB server dictionaries and modules. The Oracle Berkeley DB XML database provides a storage backend for the administration meta-data. The server-side scripts are developed in Ruby programming language.

All the data about users, dictionaries, permissions and other control data are stored in the DB XML database in the XML format. Each dictionary module of the DEB server uses a common interface to access data from this administration database.

The administration module provides several services – user authentication, access rights control, entry locking and journaling of dictionary changes.


user management - Mozilla Firefox

File Edit View History Bookmarks Tools Help

https://[redacted]/admin?action=listusers&edituser=adam

login	name / email	organization / address	password / comment
adam	Adam Rambousek / xrambous@fi.muni.cz	Faculty of Informatics / Botanická 68a Brno	

cpa ☐ · * cpa · debdict ☑ · * cia r · cod11 · diderot r · gslov · ode · ote · scfin r · scfis r · scs
debvisdic · * diderot · scfin · scfis · cs · wne
wngre · wnh · wnpol · wrus
praled ☐
* cia · diderot · gslov · praled · ssc · ssjc
admin ☐

save · new pass · del user

all users (169)
abican (Aleš Bičan) acerna (Anna Černá) … ti (Ágnes Gya… … cova (Agáta Karčová)
anneu (Anne Urbschat) aoravcova (Adri… …n) arum (An… …á) asufliar (Alexandra
Šufliarska) bchocholova (Bronislava Choc… …) birkhahn… …na Bozděchová, CSc.)
chudomelova (Věra Chudomelová) corn… cpa02 (CPA… …05 (CPA05) cpa06 (CPA06)
cpa07 (CPA07) cpa08 (CPA08) cpa09 (… …roup) dfiser… majeurakova (Daniela
Majeuráková) duzi (Marie Duží) dvasilis… …kova (Dáša Z… …rjavec (Tomaž Erjavec)
fcermak (František Čermák) fcvrcek (Fra… …zgerald) fsti… …ajdošová) gazi (Gabriel Gaži)
gkotze (Gideon Jozua Kotzé) golanova (Hana Goláňová) gryciuk (Joanna Gryciuk) gszarv… …orikova (Helena Ivoríková)
hkonecna (Hana Konečná) hlava (Jaroslava Hlaváčová) hlavsem (skupinový účet do 1.10.200… …va (Hana Palová) hradec (Jan
Hradec) hruskova (Jitka Hrušková) inemcova (Ivana Němcová) isvobodova (Ivana Svobod… …n Bosák) ichromy (Jan Chromý)

Figure 2.13: User management showing how access rights modify the dictionary list in DEBDict; list for selected user is on the left, list of all dictionaries is on the right.

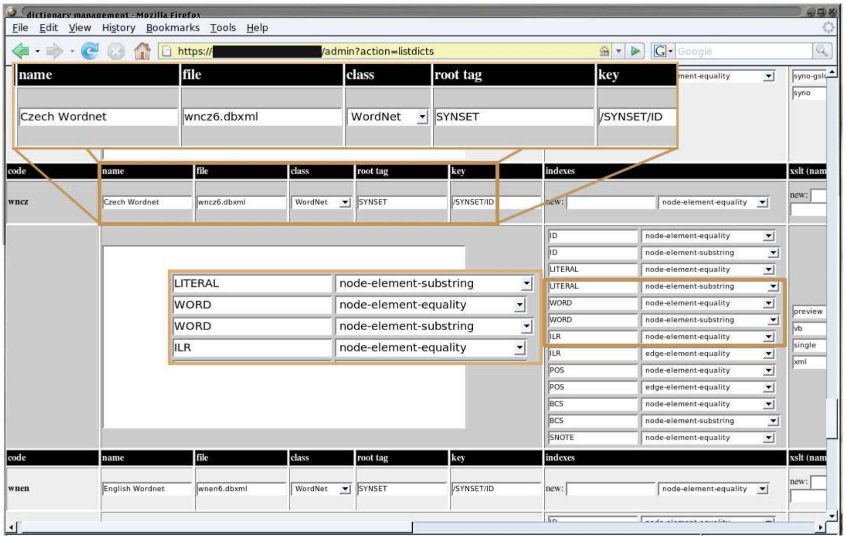
The administration interface is a web-based application where the web pages are generated using an HTTP template which allows easy design and content modification and then served to the users by a light-weight web server – WEBrick [San04]. The users are authenticated using standard HTTP authentication mechanism. The administration module extends the standard interface for passwords stored in a file and loads user's login and password from the XML database. Each change in user accounts or access rights is propagated to all DEB services in the real-time.

2.3.3.2 The Dictionary Management

For each dictionary, the administrator has to define several attributes (see the Figure 2.15). The minimal set of attributes contains a unique dictionary code, a database filename and a dictionary class (the implementation class in Ruby), the other attributes are more or less optional.

```
<user>
  <login>adam</login>
  <name>Adam Rambousek</name>
  <email>xrambous@fi.muni.cz</email>
  <org>Faculty of Informatics</org>
  <addr>Botanicka 68a, Brno</addr>
  <pass>3Ja8ivX12OB0U</pass>
  <services><service code="debdict">
    <dict code="scs" perm="r"/>
    <dict code="scfis" perm="r"/>
    <dict code="cia" perm="r"/>
    <dict code="scfin" perm="r"/>
    <dict code="diderot" perm="r"/>
  </service></services>
</user>
```

Figure 2.14: XML entry for the user from the Figure 2.13.


dictionary management - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Google

https://[redacted]/admin?action=listdicts

code	name	file	class	root tag	key	indexes	xslt (nam
wncz	Czech Wordnet	wncz6.dbxml	WordNet	SYNSET	/SYNSET/ID	new: node-element-equality	new:
wnen	English Wordnet	wnen6.dbxml	WordNet	SYNSET	/SYNSET/ID	new: node-element-equality	new:

ID	node-element-equality
ID	node-element-substring
LITERAL	node-element-equality
LITERAL	node-element-substring
WORD	node-element-equality
WORD	node-element-substring
ILR	node-element-equality
ILR	edge-element-equality
POS	node-element-equality
POS	edge-element-equality
BCS	node-element-equality
BCS	node-element-substring
SNOTE	node-element-equality

syno-gsl, syno, preview, vb, single, xml

Figure 2.15: Dictionary management showing basic information and indexed elements for the Czech WordNet dictionary.

The meaning of the dictionary attributes is:

- the dictionary name is displayed to users by the client application.
- the definition of the XML entry root tag and its key element are needed for XML import and for searching (in case, the application does not have its own, more complex search method).
- indexes speed up search operations, so each element or attribute that is used in user queries should be indexed.
- XSLT templates transform XML data to another form suitable for presentation or machine processing.

Extra dictionary attributes are required for the WordNet dictionaries:

- each WordNet dictionary is linked to the client software by the client package code.
- the WordNet Dictionaries can refer to each other using the specified "equivalence tags."
- in the next field, the administrator can list dictionaries that should be reloaded after an edit action in the client (usually in another dictionary).
- and the last option specifies related dictionaries – for example, several national WordNets linked with ILI (Inter-Lingual Index). It is possible to display the same entry in different languages or to copy entries between languages.

2.3.3.3 Import and Export

The import function takes an XML file and stores the data into the DB XML database. The XML file has to be uploaded to the server (it is possible to upload it through web interface). All entries must share the same root tag (specified in the dictionary management), entries with different root tags are ignored. The administrator can choose if he or she wants to delete all the entries from database before the import or just add the new entries. The import utilizes two methods for XML reading. The first method loads the whole XML file into memory and uses an XML parser on the big document. This method is accurate, unfortunately it has exponential time complexity, so it can take hours for large XML files (over 10 MB). The second method uses regular expressions to read entries one by one from the XML file and then each single entry is parsed. Entries are stored in the database with value of the specified key tag as a unique key. The administrator is informed about the import progress on the web page – a number of processed entries, a total number of entries, an estimated time till the end and last ten entry keys are displayed.

The administration module also supports export from database to plain XML file, the output files may be compressed to save disk space. The export also has an option to save the file in the form of a Ruby language script that will setup the database and import initial data. This is needed for the administration database itself. The output files are saved in a specified directory on the server and the administrator is informed about the export progress. Once the export ends, the administrator is offered a link to download the file through the web interface. The same function is used also for daily database backup.

2.3.3.4 Locking and Sequences of Identifiers

The administration interface offers entry locking management to other DEB server modules. If multiple users can edit the database at the same time (which is one of the basic advantages of the client-server architecture), it is crucial to provide exclusive write locking of entries so that two users are not able to edit the same entry at a time. Decisions about entry locking depends on each application design:

1) when should an entry be locked and unlocked?
2) should only the edited entry be locked or should the locking affect other entries too?

An application then sends the request to the administration module which updates the lock database. The administration module provides several functions – besides simple lock and unlock functions, it can tell which user has locked a given entry, return the list of locks for selected user and/or dictionary or group several locks together if they are related. The administrator has access to the list of all locks and he or she can also delete chosen locks if the application did not release them correctly.

Newly created entries should have a unique identifier. If the application does not generate its own identifiers, the administration module can provide such service. It is possible to set an identifier pattern for each dictionary – this pattern looks like `CZE-[id]` and `[id]` will be replaced with sequentially increased number. The administrator can also affect the number used.

2.3.3.5 The Installation Packages

The administration interface supports automated creation of Firefox Extension installation packages (XPI). If the administrator specifies a Relax NG schema for the dictionary, it is possible to automatically transform this schema to an application design description in the XUL description language and the supporting code in JavaScript. The application created in this way supports basic forms – single and multiple text fields, select-boxes of specific values or relational links to other dictionaries. It can serve as basis for custom modifications. Of course, the application is able to connect to server, load data from server and save a modified entry back. We are currently working on more complex support for creation of new packages, mainly for the DEBVisDic client packages.

2.3.4 How To Make a Sample Dictionary

2.3.4.1 New Dictionary Definition

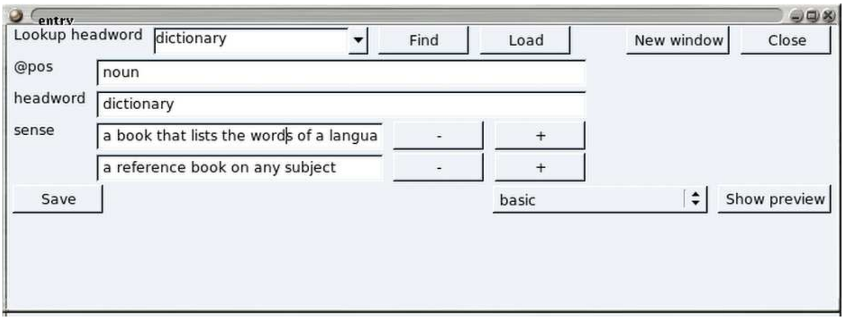
As a first step, the administrator needs to provide basic information about the dictionary. The dictionary data can be loaded from an XML file or it can be built from scratch. The administrator must specify an entry root element, the XPath specification of a unique key, several indexes for fast querying and an XML schema of the entry.

Let us create a demonstration dictionary from scratch, we will name the root element `entry` and have the unique key identifier in the element `/entry/headword`. The corresponding Relax NG schema is given in the Figure 2.16.

This schema describes entry with one `headword` element, with `pos` attribute, and one or more `sense` elements. Of course, Relax NG supports description of much more complex XML structures.

```
<element name="entry">
  <element name="headword">
    <attribute name="pos"> <text/> </attribute>
    <text/>
  </element>
  <oneOrMore>
    <element name="sense"> <text/> </element>
  </oneOrMore>
</element>
```

Figure 2.16: A part of the Relax NG dictionary schema.



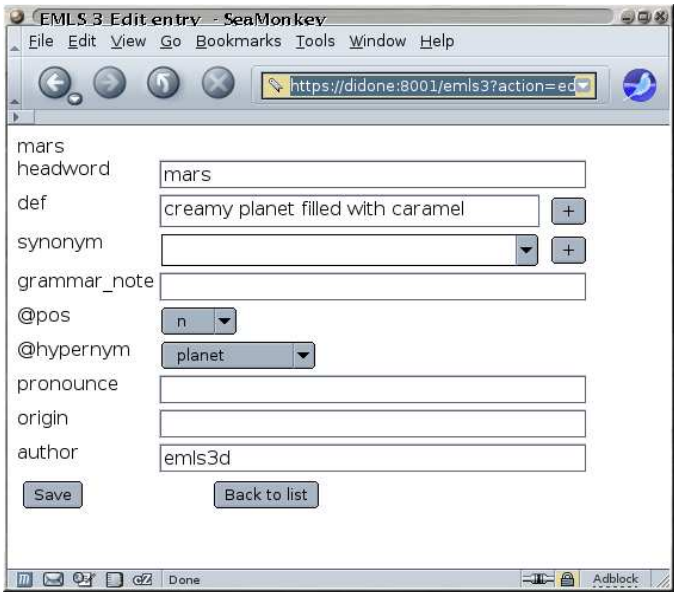
Lookup headword: dictionary
@pos: noun
headword: dictionary
sense: a book that lists the words of a langua
a reference book on any subject
basic

Figure 2.17: An example client application generated by the DEB administration interface according to the dictionary schema from the Figure 2.16.

2.3.4.2 Preparation of an Installation Package

The preparation of a new basic client application package requires selection of a dictionary and running the package generation function. The administration module checks the Relax NG schema and finds all elements or attributes that contain the `text` child element. All such elements and attributes are transformed to XUL textbox fields with the respective name as a label describing the field. If an element can occur multiple times in the entry (such as the `sense` tag in our example), buttons for adding and removing the textbox are added to the application form, too.

The created JavaScript supports loading and saving documents and


EMLS 3 Edit entry - SeaMonkey
File Edit View Go Bookmarks Tools Window Help
https://didone:8001/emls3?action=ed
Save
Back to list
Done
Adblock

mars

headword	mars
def	creamy planet filled with caramel
synonym	
grammar_note	
@pos	n
@hypernym	planet
pronounce	
origin	
author	emls3d

Figure 2.18: A web service automatically built following a dictionary schema

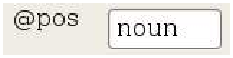
also searching for documents. The application thus enables querying each indexed field specified in the dictionary management interface. For example, users can easily find all nouns.

All the created application files are then packaged into the Firefox extension installation package (XPI). Users can download this package for installation or individual files for editing. An example of the resulting application is shown on the Figure 2.17.

For the new client, there are also two basic preview templates (in XSLT) saved on the server side. One provides basic entry preview displaying all the data and the second displays raw XML data.

For certain environments that either do not allow users to install new software packages or where the deployment of the software would be too time consuming, the DEB II server is able to generate simple web-service (see the Figure 2.18). The same as for XPI package generation, this function uses the dictionary Relax NG schema and generates a XUL form for

a)
```
<textbox id="entry.headword.@pos"/>
```



b)
```
<menulist id="entry.headword.@pos">
  <menupopup>
    <menuitem label="noun"/>
    <menuitem label="verb"/>
    <menuitem label="adjective"/>
    <menuitem label="adverb"/>
  </menupopup>
</menulist>
```

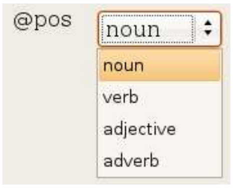


Figure 2.19: Change of a textbox field to a drop-down list.

remote access. To work with the dictionary, a user needs a web browser based on the Mozilla engine (Firefox, SeaMonkey, Netscape, Camino, ...). All parts of the generated web-service are easily customizable via XSLT templates.

2.3.4.3 Application Customization

Thanks to the design of applications based on the Mozilla development platform, these applications are easily customizable.

Any change in the layout and design of the form is done by editing the XUL (XML User-interface Language) files accompanied with standard CSS stylesheets. The application logic (i.e. procedures implemented in JavaScript) stays the same for a new layout. Combination of XUL and CSS languages is very powerful and supports long list of features that are commonly used in desktop applications. To give a simple example, we can show how to change the Part-Of-Speech textbox field into a drop-down list, see the Figure 2.19.

The application localization (translation of the user interface to another language) is one of the core features of the Mozilla XPI packages. As we can see in the example, the field labels contain XML element names only (marked with `&`). This allows the application designer to change them to general textual labels that are more informative to users. The actual texts are stored in a DTD (Document Type Definition) file as XML entity definitions, where they can be adjusted to any texts in one place. For the sake of the above mentioned localization of the application, it is possible to include several DTD files for different languages

application.xul:	`<label value="&entry.headword;"/>`
en-US/application.dtd:	`<!ENTITY entry.headword "headword">`
cs-CZ/application.dtd:	`<!ENTITY entry.headword "heslo">`

Figure 2.20: A field label and the respective entity in the localized DTD files.

into the installation package and (automatically) switch between them (see the Figure 2.20).

After all the application source files are modified to meet the designer's requirements, he or she can upload them using the administration interface and let it build a new version of the installation package.

The application designer can also supplement the dictionary editor with more preview templates or modify the existing ones for different data presentation. When adding a new template, the template name must be added to the dictionary description in the database management interface. The modified templates are again uploaded to the server via the administration interface.

2.3.5 Usage Variability – The Users' Interfaces

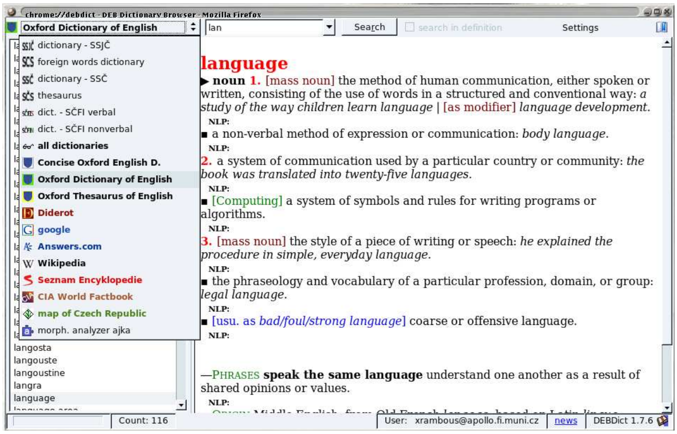
The DEB clients are written in XUL and JavaScript and integrate with Mozilla Firefox web browser. This allows the developers to use both Mozilla's user interface engine and its HTML/XHTML rendering engine as well as built-in components for interaction with filesystem on client computers, XPath interpreter, RDF processor etc.

Due to the feature-rich client architecture the developers may decide whether certain operations should be done on the server or on client parts – e.g. XSLT transformation can be done on both sides.

The particular DEB clients that are currently being implemented within the DEB platform include:

- DEBVisDic – complete new version of the successful WordNet semantic network editor and browser VisDic, see the Section 2.3.5.2.

- DEBDict – a general dictionary browser.



language
▶ **noun** 1. [mass noun] the method of human communication, either spoken or written, consisting of the use of words in a structured and conventional way: *a study of the way children learn language* | [as modifier] *language development.*
NLP:
■ a non-verbal method of expression or communication: *body language.*
NLP:
2. a system of communication used by a particular country or community: *the book was translated into twenty-five languages.*
NLP:
■ [Computing] a system of symbols and rules for writing programs or algorithms.
NLP:
3. [mass noun] the style of a piece of writing or speech: *he explained the procedure in simple, everyday language.*
NLP:
■ the phraseology and vocabulary of a particular profession, domain, or group: *legal language.*
NLP:
■ [usu. as *bad/foul/strong language*] coarse or offensive language.
NLP:

—PHRASES **speak the same language** understand one another as a result of shared opinions or values.
NLP:

Figure 2.21: The DEBDict common interface to several dictionaries with different structures.

- Czech Onomastic Dictionary – newly prepared dictionary of Czech proper names and their origins
- PRALED – implemented in cooperation with the Institute of Czech Language, Czech Academy of Sciences in Prague. The task with PRALED is to build a new Czech Lexical Database. See the Section 2.3.5.4 for more information.
- DEB CPA – tool used for several distant teams to work on a new resource of *Corpus Pattern Analysis.* It is further described in the Section 2.3.5.3.
- DEB TEDI – the main tool for preparation of a new terminological dictionary of Czech art terms. This work is a joint project of the Faculty of Fine Arts, Brno University of Technology and Masaryk University. The aim of the project is to build a terminological database consisting of about 5 000 dictionary entries which are classified into categories and supplemented with term definitions, translations info English, German and French, and with Czech usage examples. The resulting dictionary will be offered as a publicly

available application directed especially to fine arts students.

- Cornetto – several client tools working over a complex database schema of three interconnected dictionaries of Dutch. Prepared in cooperation with the University of Amsterdam. Details are presented in the Section 2.3.5.5.

- VisualBrowser – the DEB II client-server architecture allows an easy connection of other existing applications to the DEB server. An example of such application is a direct interface to the VisualBrowser tool [Nev05] that now displays the graphical representation of relations between elements stored in various DEB server databases.

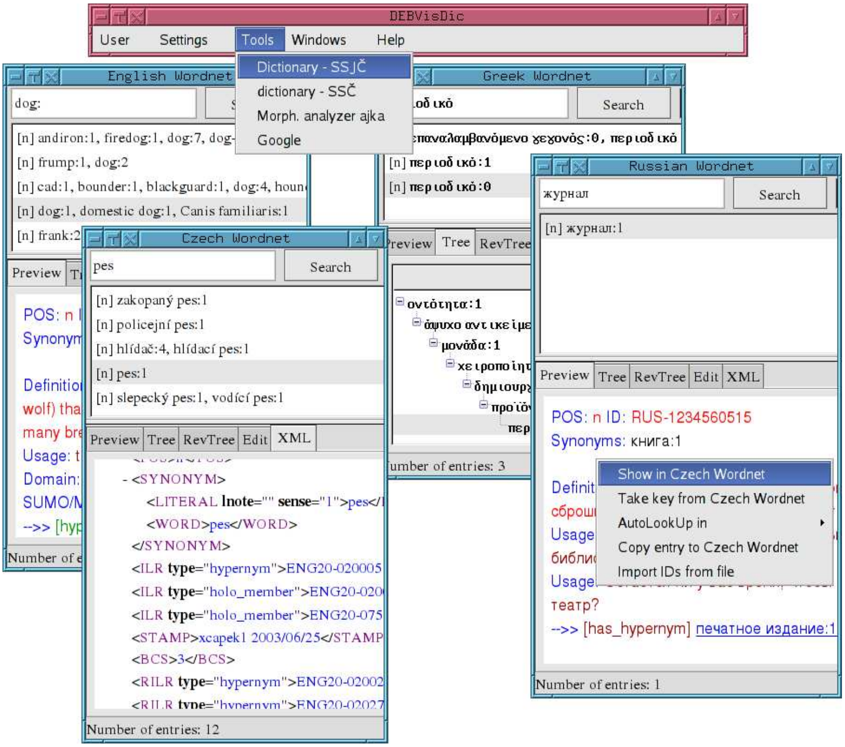
2.3.5.1 DEBDict

The dictionary browser DEBDict originally started as a demonstration application for the DEB platform and it gradually evolved into an indispensable resource for several hundreds of linguists in several countries. The DEBDict functionalities include:

- multilingual user interface (English and Czech, others can be easily added)
- queries to several XML dictionaries (of different underlying structure) with the result passed through an XSLT transformation
- integration with Czech morphological analyzer
- connection to an external website
- connection to the GRASS geographical information system (display of geographical links directly on their positions within a cartographic map) or any similar application

The version of DEBDict that is currently running on the MU NLP Centre server provides a common interface to 7 dictionaries (see the Figure 2.21):

- the Dictionary of Literary Czech Language (SSJČ [P+02], 180 000 entries)
- the Reference Dictionary of Czech Language (PSJČ [Hav57], with 200 000 entries)
- the Dictionary of Literary Czech (SSČ [F+95], 49 000 entries)
- the Dictionary of foreign words (SCS [KP+99], 46 000 entries)
- the Dictionary of Czech Synonyms (thesaurus [PJ94], 23 000 entries)



dog:

[n] andiron:1, firedog:1, dog:7, dog-

[n] frump:1, dog:2

[n] cad:1, bounder:1, blackguard:1, dog:4, houn

[n] dog:1, domestic dog:1, Canis familiaris:1

[n] frank:2

POS: n I
Synonym
Definitio
wolf) tha
many bre
Usage: t
Domain:
SUMO/M
-->> [hyp

Number of e

pes

[n] zakopaný pes:1

[n] policejní pes:1

[n] hlídač:4, hlídací pes:1

[n] pes:1

[n] slepecký pes:1, vodící pes:1

```
  - <SYNONYM>
      <LITERAL lnote="" sense="1">pes</l
      <WORD>pes</WORD>
    </SYNONYM>
    <ILR type="hypernym">ENG20-020005
    <ILR type="holo_member">ENG20-020
    <ILR type="holo_member">ENG20-075
    <STAMP>xcapek1 2003/06/25</STAMP
    <BCS>3</BCS>
    <RILR type="hypernym">ENG20-02002
    <RILR type="hypernym">ENG20-02027
```

Number of entries: 12

περιοδικό

επαναλαμβανόμενο χεχονός:0, περιοδικό

[n] περιοδικό:1

[n] περιοδικό:0

οντότητα:1
άψυχο αντικείμε
μονάδα:1
χειροποίητ
δημιουργ
προϊόν
περ

umber of entries: 3

журнал

[n] журнал:1

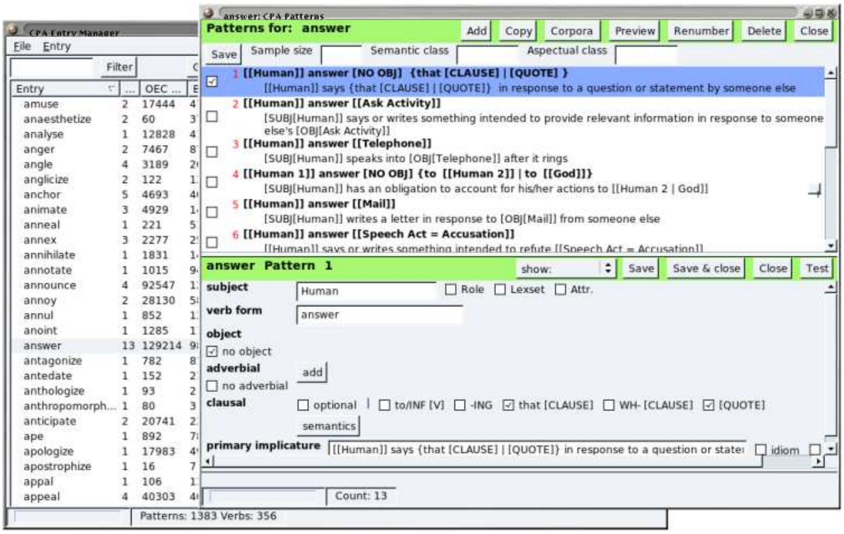
POS: n ID: RUS-1234560515
Synonyms: книга:1
Definit
сброшı
Usage
библис
Usage
театр?
-->> [has_hypernym] печатное издание:1

Number of entries: 1

Figure 2.22: The DEBVisDic main interface

- two parts of the Dictionary of Czech Phrasal Words and Idioms (shortly SČFI [Č+83], 4 000 verbal entries and 10 000 non-verbal entries)
- the Diderot encyclopedia (90 000 entries)

As an addition, DEBDict features an interconnection to several web systems (such as Google or Answers.com) and the GRASS geographical system with the list of the Czech towns and cities.



Entry	...	OEC ...	E
amuse	2	17444	4
anaesthetize	2	60	3
analyse	1	12828	4
anger	2	7467	8
angle	4	3189	2
anglicize	2	122	1
anchor	5	4693	4
animate	3	4929	1
anneal	1	221	5
annex	3	2277	2
annihilate	1	1831	1
annotate	1	1015	9
announce	4	92547	1
annoy	2	28130	5
annul	1	852	1
anoint	1	1285	1
answer	13	129214	9
antagonize	1	782	8
antedate	1	152	2
anthologize	1	93	2
anthropomorph...	1	80	3
anticipate	2	20741	2
ape	1	892	7
apologize	1	17983	4
apostrophize	1	16	7
appal	1	106	1
appeal	4	40303	4

Patterns: 1383 Verbs: 356

Patterns for: answer

[✓] 1 **[[Human]] answer [NO OBJ] {that [CLAUSE] | [QUOTE] }**
[[Human]] says {that [CLAUSE] | [QUOTE]} in response to a question or statement by someone else

☐ 2 **[[Human]] answer [[Ask Activity]]**
[SUBJ[Human]] says or writes something intended to provide relevant information in response to someone else's [OBJ[Ask Activity]]

☐ 3 **[[Human]] answer [[Telephone]]**
[SUBJ[Human]] speaks into [OBJ[Telephone]] after it rings

☐ 4 **[[Human 1]] answer [NO OBJ] {to [[Human 2]] | to [[God]]}**
[SUBJ[Human]] has an obligation to account for his/her actions to [[Human 2 | God]]

☐ 5 **[[Human]] answer [[Mail]]**
[SUBJ[Human]] writes a letter in response to [OBJ[Mail]] from someone else

☐ 6 **[[Human]] answer [[Speech Act = Accusation]]**
[[Human]] says or writes something intended to refute [[Speech Act = Accusation]]

answer Pattern 1

subject Human ☐ Role ☐ Lexset ☐ Attr.
verb form answer
object
[✓] no object
adverbial
☐ no adverbial
clausal ☐ optional | ☐ to/INF [V] ☐ -ING [✓] that [CLAUSE] ☐ WH- [CLAUSE] [✓] [QUOTE]
primary implicature [[Human]] says {that [CLAUSE] | [QUOTE]} in response to a question or state ☐ idiom ☐

Count: 13

Figure 2.23: The DEB CPA tool.

2.3.5.2 DEBVisDic

DEBVisDic was one of the first applications built over the DEB II platform – it was designed as a completely new client-server tool for WordNet browsing and editing that should serve as a successor of the VisDic program presented in the Section 2.2.

DEBVisDic uses new versatile interface (see the Figure 2.22 that allows the user to arrange the work without any limitations. Of course, DEBVisDic contains all the main features that were present in VisDic:

- multiple views of multiple WordNets
- freely defined text views
- synset editing
- hypero-hyponymic tree
- query result lists
- plain XML view of a synset
- synchronization
- inter-dictionary linking

- tree browsing
- consistency checks
- journaling
- user configuration

With the help of the DEB platform reusability, DEBVisDic will be supplemented with many new features that are currently accessible only as separate tools or resources. This functionality includes:

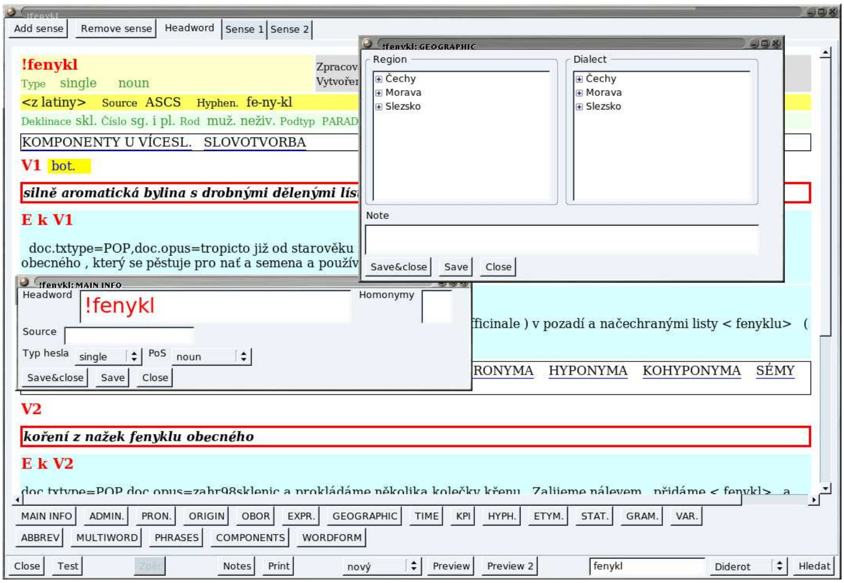
- connection to a morphological analyzer (for languages, where it is available)
- connection to language corpora, including Word Sketches statistics
- access to any electronic dictionaries stored within the DEB server
- searching for literals within encyclopedic web sites
- and many others

Currently, DEBVisDic is also used for preparation of new Polish, Hungarian, Slovenian, Dutch (within the Cornetto project) and Afrikaans WordNets and it is proposed as the main tool for the prepared Global WordNet Grid.

2.3.5.3 DEB CPA Editor and Browser

Corpus Pattern Analysis (CPA, [Han04]) is a new technique for mapping meaning to words in text. No attempt is made in CPA to identify the meaning of a verb or noun directly, as a word in isolation. Instead, meanings are associated with prototypical sentence contexts. Concordance lines are grouped into semantically motivated syntagmatic patterns. Associating a “meaning” with each pattern is a secondary step, carried out in close coordination with the assignment of concordance lines to patterns.

CPA editing tool (see the Figure 2.23) displays the list of verb entries, along with the information who and when updated the entry. Each entry consists of several patterns (the number of patterns is not limited) and it is possible to freely modify their order and content. The main part of the tool, the pattern editing window, allows to enter and modify all the information about one pattern. The form is very versatile, e.g. it allows to add any number of subject/object alternations. The tool is connected to an on-line resource – it is possible to look up subject and object semantic type in Brandeis Semantic Ontology [PHL$^+$06] which is hosted on a web



!fenykl
Zpracov
Vytvořen
Type single noun
<z latiny> Source ASCS Hyphen. fe-ny-kl
Deklinace skl. Číslo sg. i pl. Rod muž. neživ. Podtyp PARAD
KOMPONENTY U VÍCESL. SLOVOTVORBA
V1 bot.
silně aromatická bylina s drobnými dělenými lís
E k V1
doc.txtype=POP,doc.opus=tropicto již od starověku
obecného , který se pěstuje pro nať a semena a použív
ficinale) v pozadí a načechranými listy < fenyklu> (
RONYMA HYPONYMA KOHYPONYMA SÉMY
V2
koření z nažek fenyklu obecného
E k V2
doc.txtype=POP doc.opus=zahr98sklenic a prokládáme několika kolečky křenu Zalijeme nálevem přidáme < fenykl> a

!fenykl: GEOGRAPHIC
Region: Čechy, Morava, Slezsko
Dialect: Čechy, Morava, Slezsko
Note

!fenykl: MAIN INFO
Headword: !fenykl
Homonymy
Source
Typ hesla: single
PoS: noun

nový
fenykl
Diderot

Figure 2.24: The PRALED user interface

server at Brandeis University. Examples documenting the pattern are taken from BNC using a modified version of Bonito2 corpus manager that is integrated to the DEB CPA tool.

2.3.5.4 The PRALED Lexicographic Station

This client is designed for the development of the Czech Lexical Database (CLD, denoted also as LEXIKON 21 [RK07]) and it serves as a main tool in preparation of the new comprehensive and exhaustive database of lexicographic information for the Czech language. The user's part of the PRALED tool is presently under the development in the Institute of Czech Language (ICL), Czech Academy of Sciences, Prague.

The PRALED system offers the following functionality:

- queries to several XML dictionaries (of different underlying structures), particularly to all relevant Czech dictionaries, i.e. SSJČ, SSČ, SCS, SČFI and DIDEROT (see [P$^+$02, F$^+$95, KP$^+$99, Č$^+$83]),

- editing existing or writing new dictionary entries. A lexicographer can use a set of forms which define the structure of the entry and fill in all relevant fields (see the Figure 2.24) which presently are:
  - pronunciation and spelling
  - morphological properties (POS, the respective grammatical categories)
  - description of the meaning (entry definition)
  - word formation nest (subnet)
  - syntactic properties (most often valencies)
  - stylistic, domain and regional features
  - semantic relations to other entries (cross-references)
  - etymological information
  - integration with Czech morphological analyzer
  - connection to an external website (Google, or Answers.com)
  - remarks and additional comments
  - integration with the corpus manager Bonito2 and Word Sketch Engine [KRST04], which allows a lexicographer to obtain the sorted individual word contexts including frequencies and statistical distribution parameters (salience).

2.3.5.5 Cornetto

The Cornetto project is a Dutch national project funded in the STEVIN framework. The goal of this project is to build a lexical semantic database for Dutch, covering 40K entries, including the most generic and central part of the language. Cornetto will combine the structures of both the Princeton WordNet and FrameNet for English [FBS04], by combining and aligning two existing semantic resources for Dutch: the Dutch WordNet [Vos98] and the Referentie Bestand Nederlands [MMdM99]. The Dutch WordNet (DWN) is similar to the Princeton WordNet for English, and the Referentie Bestand (RBN) includes frame-like information as in FrameNet plus additional information on the combinatoric behaviour of words in a particular meaning. The combination of the two lexical resources will result in a much richer relational database that may improve natural language processing (NLP) technologies, such as word sense-disambiguation and language-generation systems. In addition to

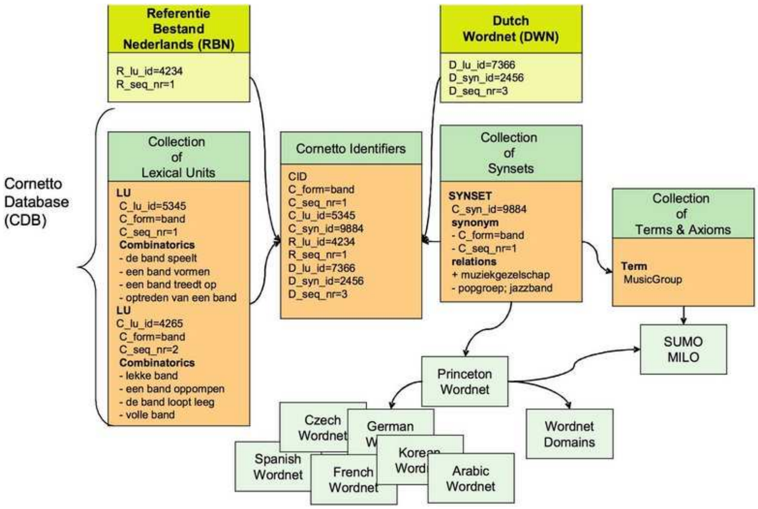


Figure 2.25: Data collections in the Cornetto database (the schema was prepared by Piek Vossen).

merging the WordNet and FrameNet-like information, the database is also mapped to a formal ontology to provide a more solid semantic backbone.

The prepared Cornetto database (CDB) consists of three main data collections:

1. collection of Lexical Units, mainly derived from the RBN
2. collection of Synsets, mainly derived from DWN
3. collection of Terms and axioms, mainly derived from SUMO [NP01] and MILO [NT04]

In addition to the three data collections, a separate table of so-called Cornetto Identifiers (CIDs) is provided. These identifiers contain the relations between the lexical units and the synsets in the CDB but also to the original word senses and synsets in the RBN and DWN. The Figure 2.25 shows an overview of the different data structures and their relations.

Since one of the basic parts of the Cornetto database is the Dutch WordNet, we have decided to use DEBVisDic as the core for Cornetto client software. We have developed four new modules, described in more details below. All the databases are linked together and also to external resources (Princeton English WordNet and SUMO ontology), thus every possible user action had to be very carefully analyzed and described.

Cornetto Lexical Units The Cornetto foundation is formed by Lexical Units, so let us describe their client package first. Each entry contains complex information about morphology, syntax, semantics and pragmatics, and also lots of examples with complex substructure. Thus one of the important tasks was to design a preview to display everything needed by the lexicographers without the necessity to scroll a lot. The examples were moved to separate tab and only their short resumé stayed on the main preview tab.

Lexical units also contain semantic information from RBN that cannot be published freely because of licensing issues. Thus DEBVisDic here needs to differentiate the preview content based on the actual user's access rights.

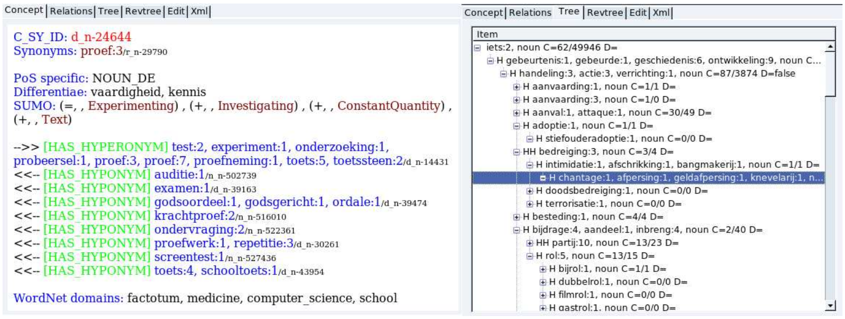
The same ergonomic problem had to be resolved in the edit form. The whole form is divided to smaller groups of related fields (e.g. morphology) and it is possible to hide or display each group separately. By default, only the most important parts are displayed and the rest is hidden.

Another new feature developed for Cornetto is the option to split the edited entry. Basically, this function copies all content of edited entry to a new one. This way, users may easily create two lexical units that differ only in some selected details.

Because of the links between all the data collections, every change in lexical units has to be propagated to Cornetto Synsets and Identifiers. For example, when deleting a lexical unit, the corresponding synonym must be automatically removed from the synset dictionary.

Cornetto Synsets Synsets are even more complex than lexical units, because they contain lots of links to different sources – links to lexical units, relations to other synsets, equivalence links to Princeton English WordNet, and links to the ontology.

Again, designing the user-friendly preview containing all the information was very important. Even here, we had to split the preview to two tabs – the first with the synonyms, domains, ontology, definition and


Concept | Relations | Tree | Revtree | Edit | Xml |
Concept | Relations Tree | Revtree | Edit | Xml |
Item

```
C_SY_ID: d_n-24644
Synonyms: proef:3/r_n-29790

PoS specific: NOUN_DE
Differentiae: vaardigheid, kennis
SUMO: (=, , Experimenting) , (+, , Investigating) , (+, , ConstantQuantity) ,
(+, , Text)

-->> [HAS_HYPERONYM] test:2, experiment:1, onderzoeking:1,
probeersel:1, proef:3, proef:7, proefneming:1, toets:5, toetssteen:2/d_n-14431
<<-- [HAS_HYPONYM] auditie:1/n_n-502739
<<-- [HAS_HYPONYM] examen:1/d_n-39163
<<-- [HAS_HYPONYM] godsoordeel:1, godsgericht:1, ordale:1/d_n-39474
<<-- [HAS_HYPONYM] krachtproef:2/n_n-516010
<<-- [HAS_HYPONYM] ondervraging:2/n_n-522361
<<-- [HAS_HYPONYM] proefwerk:1, repetitie:3/d_n-30261
<<-- [HAS_HYPONYM] screentest:1/n_n-527436
<<-- [HAS_HYPONYM] toets:4, schooltoets:1/d_n-43954

WordNet domains: factotum, medicine, computer_science, school
```

```
iets:2, noun C=62/49946 D=
  H gebeurtenis:1, gebeurde:1, geschiedenis:6, ontwikkeling:9, noun C...
    H handeling:3, actie:3, verrichting:1, noun C=87/3874 D=false
      H aanvaarding:1, noun C=1/1 D=
      H aanvaarding:3, noun C=1/0 D=
      H aanval:1, attaque:1, noun C=30/49 D=
      H adoptie:1, noun C=1/1 D=
        H stiefouderadoptie:1, noun C=0/0 D=
      HH bedreiging:3, noun C=3/4 D=
        H intimidatie:1, afschrikking:1, bangmakerij:1, noun C=1/1 D=
          H chantage:1, afpersing:1, geldafpersing:1, knevelarij:1, n...
        H doodsbedreiging:1, noun C=0/0 D=
        H terrorisatie:1, noun C=0/0 D=
      H besteding:1, noun C=4/4 D=
      H bijdrage:4, aandeel:1, inbreng:4, noun C=2/40 D=
        HH partij:10, noun C=13/23 D=
        H rol:5, noun C=13/15 D=
          H bijrol:1, noun C=1/1 D=
          H dubbelrol:1, noun C=0/0 D=
          H filmrol:1, noun C=0/0 D=
          H gastrol:1, noun C=0/0 D=
```

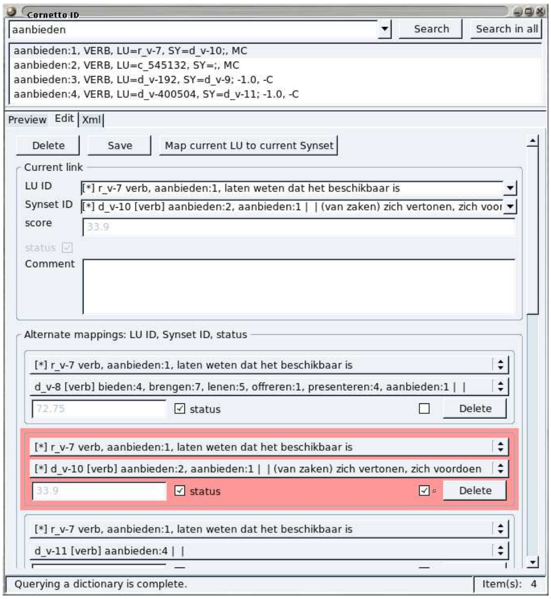
Figure 2.26: Cornetto Synsets window, showing a preview and a hyperonymy tree

short representation of internal relations, and the second with full information on each relation (both internal and external to English WordNet). Each link in the preview is clickable and displays the selected entry in the corresponding dictionary window (for example, clicking on a synonym opens a lexical unit preview in the lexical unit window, see the Figure 2.26).

The synset window offers also a tree view representing a hypernym/hyponym tree. Since the hypero/hyponymic hierarchy in WordNet forms not a simple tree but a directed graph, another tab provides the reversed tree displaying links in the opposite direction (this concept was introduced in the VisDic WordNet editor). The tree view also contains information about each subtree's significance expressed by the number of direct hyponyms and the number of all the descendant synsets.

The synset edit form looks similar to the form in the lexical units window, with less important parts hidden by default. When adding or editing links, users may use the same queries as in dictionaries to find the right entry.

Cornetto Identifiers The lexical units and synsets are linked together using the Cornetto Identifiers (CID). For each lexical unit, the automatic aligning software has produced several mappings to different synsets (with different score values). At the very beginning, the most probable one was



aanbieden

aanbieden:1, VERB, LU=r_v-7, SY=d_v-10;, MC
aanbieden:2, VERB, LU=c_545132, SY=;, MC
aanbieden:3, VERB, LU=d_v-192, SY=d_v-9; -1.0, -C
aanbieden:4, VERB, LU=d_v-400504, SY=d_v-11; -1.0, -C

Current link
LU ID: [*] r_v-7 verb, aanbieden:1, laten weten dat het beschikbaar is
Synset ID: [*] d_v-10 [verb] aanbieden:2, aanbieden:1 | | (van zaken) zich vertonen, zich voor
score: 33.9
status ☑
Comment

Alternate mappings: LU ID, Synset ID, status

[*] r_v-7 verb, aanbieden:1, laten weten dat het beschikbaar is
d_v-8 [verb] bieden:4, brengen:7, lenen:5, offreren:1, presenteren:4, aanbieden:1 | |
72.75 ☑ status ☐ Delete

[*] r_v-7 verb, aanbieden:1, laten weten dat het beschikbaar is
[*] d_v-10 [verb] aanbieden:2, aanbieden:1 | | (van zaken) zich vertonen, zich voordoen
33.9 ☑ status ☑ Delete

[*] r_v-7 verb, aanbieden:1, laten weten dat het beschikbaar is
d_v-11 [verb] aanbieden:4 | |

Querying a dictionary is complete. Item(s): 4

Figure 2.27: Cornetto Identifiers window, showing the edit form with several alternate mappings

marked as the "selected" mapping.

In the course of work, users have several ways for confirming the automatic choice, choosing from other offered mapping, or creating an entirely new link (see the Figure 2.27). For example, a user can remove the incorrect synonym from a synset and the corresponding mapping will be marked as unselected in CID. Another option is to select one of the alternate mappings in the Cornetto Identifiers edit form. Of course, this action leads to an automatic update of synonyms.

The most convenient way to confirm or create links is to use *Map current LU to current Synset* function. This action can be run from any Cornetto client package, either by a keyboard shortcut or by clicking on the button. All the required changes are checked and carried out on the server, so the client software does not need to worry about the actual actions necessary to link the lexical unit and the synset.

Cornetto Ontology The Cornetto Ontology is based on SUMO and so is the client package. The ontology is used in synsets, as can be seen in the Figure 2.26. The synset preview shows a list of ontology relations triplets – relation type, variable and variable or ontology term.

Clicking on the ontology term opens the term preview. A user can also browse the tree representing the ontology structure.

2.4 Future Work on the Language Resources and NLP Tools

We have displayed the details of the VerbaLex verb valency frames dictionary and described the history of the design and implementation of its complex valency frames format that was needed for encapsulation of new semantic roles and links to the Czech WordNet entries. We expect the version VerbaLex 2.0 to be available by the end of 2007 with the number of 10 500 verbs fully linked to the Czech WordNet.

The nearest development of the VerbaLex dictionary includes linking of the newly added verb synsets to the Czech and Princeton WordNets. After enhancing the verb frame extraction algorithm in the syntactic parser `synt`, we are going to perform large scale checking and extending the lexicon by means of corpus text analysis.

We have described the DEB implementation platform and the main features of applications built over this framework. The DEB platform advantageously uses the client/server architecture and offers several different clients allowing to perform various lexicographic tasks. The relevant features of the DEB platform are high modularity and configurability. Thanks to them, the DEB platform represents a versatile base, on which the individual and powerful dictionary writing tools (clients) can be implemented.

The presented off-line WordNet editor and browser, VisDic, during its not so long history, proved its usefulness and contributed to the WordNet-like databases creation especially within the Balkanet project. We have shown that DEBVisDic as its successor retains its functionality and adds new functions that will allow the lexicographers and researchers to create new high quality lexical resources without which further progress in the NLP field can hardly take place.

The development of DEBVisDic is also related to current Semantic Web projects, in particular we will use the tool for building ontologies covering various domains (one of the candidates is oncology). In this connection, the above mentioned VisualBrowser [Nev05] is a suitable tool for natural presentation of a semantic network.

Even though the DEB platform is developed as open source and free platform, we believe that it offers interesting features for dictionary writing systems and that it can speed up the development in this area. With the new common administration module in the DEB development platform that is shared by all kinds of DEB client applications, the DEB platform provides a invaluable basis for new dictionary writing applications for all purposes and all types of dictionaries. The applicability of the platform is best justified with the implemented clients described in this chapter and with the 250 registered users form 14 domains/countries that currently access those applications at the Masaryk University DEB server and with 6 other server installations in Prague, Amsterdam-UvA, Amsterdam-VU, Poznan, Johannesburg and Budapest.

In the Cornetto project clients, we have presented how a combination of automatic scored strategies with the human lexicographic work can be used for merging large databases of previous dictionaries to obtain a new qualitative language resource with complex morphological, syntactic and semantic information. The presented project tools are, however, not a single purpose programs but they fit in the general framework of the DEB platform used for developing other publicly available language data tools.

With the creation of the DEB II platform and the individual clients PRALED, DEBDict, or DEBVisDic lexicographers obtain new tools, which can offer rapid development of necessary lexical resources, i.e. the Czech Lexical Database in particular and a new dictionary of Czech later. There have been no lexicographic station tools available for Czech users so far. Our final aim is to equip the DEB II platform with enough versatility for all needs, and to make it easily portable to different system installations.

Chapter 3

synt – Czech Syntax Analyzer

The main tasks of automatic natural language (NL) processing are based on a correct syntactic analysis of a NL sentence. While the quality of parsing of analytical languages (English, German, ...) has already achieved nearly satisfactory results [BBHS$^+$04, Rad93], the analysis of free word order languages still faces many problems either in the form of huge number of rules or output trees or it offers lower precision or coverage on corpus texts [HS02, JG02, Hof95].

The syntactic analysis of running texts plays a crucial role in many areas of advanced written and spoken text processing ranging from grammar checking, machine translation or phrase identification to knowledge mining and ontology acquisition. The problem of syntactic parsing is often reduced to shallow syntactic analysis, e.g. [SL00], which is sufficient for many applications where the speed of the processing is more important than obtaining an exact and deep syntactic representation of sentence. On the other hand, when the final aim is a thorough meaning representation of the input sentence, a complete parsing is inevitable. This is also the case of the `synt` system described further, which is being implemented as a part of the Normal translation algorithm of natural language sentences to constructions of Transparent intensional logic.[1]

[1] see [Hor02] and the Chapter 4

In the NLP Centre at Masaryk University, syntax parsing of the Czech language as a representative of free word order languages forms one of the main stream research tasks since the establishment of the Centre. The most advanced system out of several implemented analysers is a parser called `synt` [HK05a]. Recently, `synt` features a developed meta-grammar of Czech and a fast parsing mechanism which offers a coverage of more than 92 percent of Czech sentences[2] while keeping the analysis time on the average of 0.07s/sentence. The current development of the parser is aimed at exploring several *Best Analysis Selection* methods that allow to identify the preferred derivation tree in the output – due to the grammar ambiguity the number of possible outputs (derivation trees) in `synt` is often very high (in some cases hundreds of thousands and more). For many reasons, this number is unacceptable for further processing and so methods aimed at reducing it (while keeping the precision of the output) are being developed as well as methods that derive structural information directly from the *packed shared forest* obtained from `synt`.

The average number of parsing trees per input sentence strongly depends on the background grammar and thence on the language. There are natural language grammars producing one parsing tree but also highly ambiguous grammar systems producing enormous number of results. Ambiguity on all levels of representation is an inherent property of natural languages and it also forms a central problem of natural language parsing. A consequence of the natural language ambiguity is a high number of possible outputs of a parser that are represented by labeled trees. A grammar extracted from the Penn Treebank and tested on a set of sentences randomly generated from a probabilistic version of the grammar has in average 7.2×10^{27} parses per sentence according to Moore's work [Moo00a].

A traditional solution of these problems is presented by probabilistic parsing techniques [BN00] aiming at finding the most probable parse of a given input sentence. This methodology is usually based on the relative frequencies of occurrences of the possible relations in a representative corpus.

In the following text, we present an acquisition of training data for the best analysis selection. The underlying mechanism is based on the pruning constraints that automate the process of transformation of a de-

[2]when measured on 10 000 sentences from the DESAM corpus [PRS97]

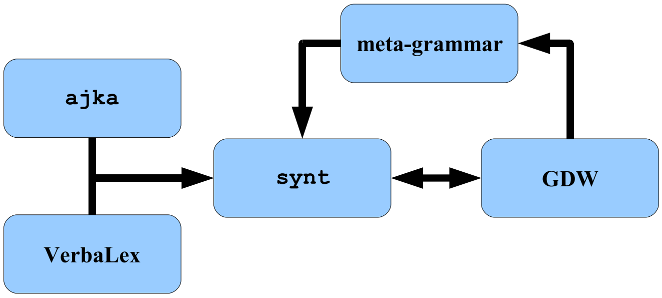


Figure 3.1: Interaction of the NLP Centre parsing tools.

pendency treebank corpus. The results of the parsing are then compared to running times of several referential parsing systems. The comparison indicates that the `synt` system is fully able to compete with the best current parsers.

This summary text on the `synt` parser is based on materials published in [HKS02, HK05b, HK05a, HHK06, KKH06, HK06, KH07, HHKK07].

The core of the syntactic parser `synt` uses the meta-grammar formalism (e.g. [Deb03, Kru03, HK05a]), which allows to specify complex sentence constituent combination rules in a maintainable way. The composition of the formalism resembles structures in Lexical functional grammars [MK91] – the meta-grammar rules are expanded within the guidance of combinatorial constructs plus contextual constraints and are supplemented with additional actions and agreement tests. The described full grammar development platform consists of the following components:

- the Czech morphological analyser `ajka` [Sed05]. This analyser provides non-disambiguated output of plain text words. The `ajka` system covers almost 400 000 Czech word lemmas, which generate over 6 million word forms.

- the VerbaLex verb valency lexicon tools, see the Section 2.1. The contextual constraints that safeguard the syntactic analysis of free word order language need extra lexical-semantic information about the sentence constituents. The creation of new verb valency lexicon which contains so called *complex valency frames* has thus become a part of the grammar development platform.

- the deep syntactic parser `synt` described in detail in the rest of this chapter. The parser development concentrates on high coverage of general corpus sentences (> 90 %). The parser is able to work with several parsing algorithms (GLR parsing, top-down, bottom-up and head-driven chart parsing). The `synt` parser also introduced a new variant of the head-driven technique, the head-driven dependent dot move algorithm (see the Section 3.2.2). This parsing technique allows to parse natural language sentences within median time of less than 0.1s/sentence.

- the user interface for linguists – the Grammar Development Workbench, GDW, is presented in the Section 3.1.1. The parser `synt` provides a command line interface, which is suitable for all forms of batch processing. However, the grammar development conducted by linguistic experts combines working with `synt` input parameters (the meta-grammar, the corpus text, parameters guiding the analysis) as well as thorough studying of `synt` outputs (tagged sentences, syntactic or dependency trees, chart graphs). All these tasks can be solved via GDW.

The grammar development platform includes several production tools and resources as described above. The Figure 3.1 shows, how these tools interact with each other. The Czech morphological analyser `ajka` and the VerbaLex verb valency lexicon provide lexical information about words or multi-word expressions in an input sentence. The syntactic parser `synt` uses the data from lexicon and the meta-grammar to create syntactic structures. The Grammar Development Workbench (GDW) is a user interface able to control the parser. Main tasks of linguistic experts working with GDW include running the parser, studying its outputs, edit and test the meta-grammar and also build a tree bank of correct derivation trees used for probability estimations and testing of grammar changes.

3.1 The Grammar Development Process

The synt meta-grammar (denoted as grammar form G1) is carefully developed by linguistic experts – currently it contains about 300 meta-rules plus special generative constructs and selectional restrictions (see the Section 3.2.1). For the actual parsing, this grammar form is automatically expanded to one of the generated forms G2 or G3 (with about 3000 or 11000 rules), which describe the context free backbone supplemented with additional tests and actions for capturing the context dependencies.

For supporting automatic consistency checks of the grammar development, a new treebank[3] of synt derivation trees is being developed. Any changes in the meta-grammar then undergo an evaluation against this treebank. All phases of this process are controlled with the Grammar Development Workbench described below, which provides all necessary functions to the linguistic experts working with the system.

The development of the parser goes hand in hand with the development of the meta-grammar format. For decisions about the order of implementation of proposed enhancement methods, different testing corpora and treebanks are used for statistical estimates of the payoff of the particular method. An example of such tree corpus utilization is presented in the Section 3.5.

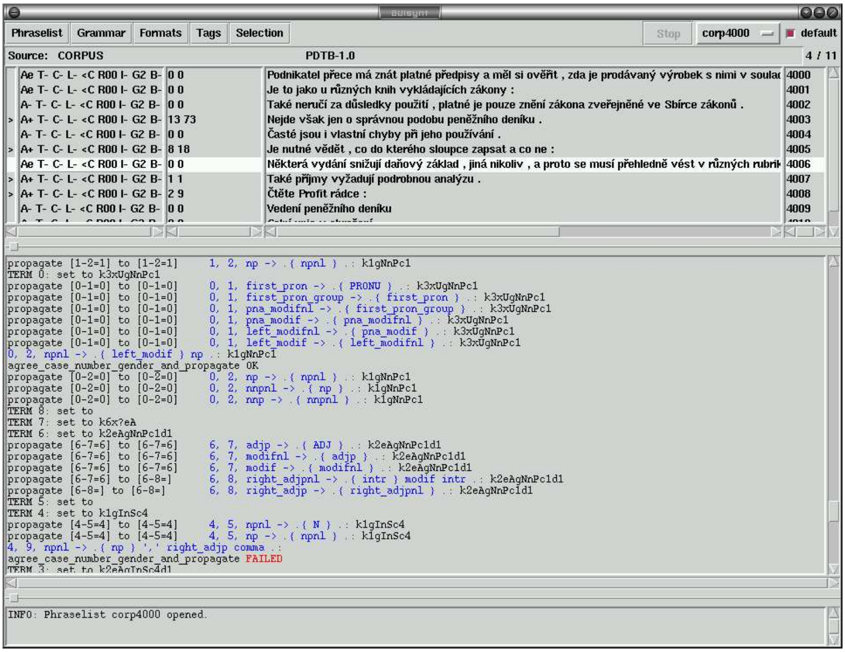
3.1.1 Grammar Development Workbench

All the above described tools and systems are presented to a user by means of a graphical front-end tool – the *Grammar Development Workbench (GDW).* Because most of the platform components (synt, ajka, ...) are controlled only from command-line, the GDW graphical user interface has been created to allow users comfortable and well arranged work with the tools.

GDW has been developed within the diploma thesis of Radek Vykydal [Vyk05] under the supervision of Aleš Horák. The system is still under development and it is now extensively tested by linguists. GDW consists of five modules:

- Gsynt – graphical user interface of the synt parser.
- TreeView – viewer for resulting syntactic trees.

[3] a collection of "trees"



Source: CORPUS PDTB-1.0 4 / 11

	Ae T- C- L- <C R00 I- G2 B-	0 0	Podnikatel přece má znát platné předpisy a měl si ověřit , zda je prodávaný výrobek s nimi v soulac	4000
	Ae T- C- L- <C R00 I- G2 B-	0 0	Je to jako u různých knih vykládajících zákony :	4001
	A- T- C- L- <C R00 I- G2 B-	0 0	Také neručí za důsledky použití , platné je pouze znění zákona zveřejněné ve Sbírce zákonů .	4002
>	A+ T- C- L- <C R00 I- G2 B-	13 73	Nejde však jen o správnou podobu peněžního deníku .	4003
	A- T- C- L- <C R00 I- G2 B-	0 0	Časté jsou i vlastní chyby při jeho používání .	4004
>	A+ T- C- L- <C R00 I- G2 B-	8 18	Je nutné vědět , co do kterého sloupce zapsat a co ne :	4005
	Ae T- C- L- <C R00 I- G2 B-	0 0	Některá vydání snižují daňový základ , jiná nikoliv , a proto se musí přehledně vést v různých rubrik	4006
>	A+ T- C- L- <C R00 I- G2 B-	1 1	Také příjmy vyžadují podrobnou analýzu .	4007
>	A+ T- C- L- <C R00 I- G2 B-	2 9	Čtěte Profit rádce :	4008
	A- T- C- L- <C R00 I- G2 B-	0 0	Vedení peněžního deníku	4009

```
propagate [1-2=1] to [1-2=1]     1, 2, np -> .{ npnl } .: k1gNnPc1
TERM 0: set to k3xUgNnPc1
propagate [0-1=0] to [0-1=0]     0, 1, first_pron -> .{ PRONU } .: k3xUgNnPc1
propagate [0-1=0] to [0-1=0]     0, 1, first_pron_group -> .{ first_pron } .: k3xUgNnPc1
propagate [0-1=0] to [0-1=0]     0, 1, pna_modifnl -> .{ first_pron_group } .: k3xUgNnPc1
propagate [0-1=0] to [0-1=0]     0, 1, pna_modif -> .{ pna_modifnl } .: k3xUgNnPc1
propagate [0-1=0] to [0-1=0]     0, 1, left_modifnl -> .{ pna_modif } .: k3xUgNnPc1
propagate [0-1=0] to [0-1=0]     0, 1, left_modif -> .{ left_modifnl } .: k3xUgNnPc1
0, 2, npnl -> .{ left_modif } np .: k1gNnPc1
agree_case_number_gender_and_propagate OK
propagate [0-2=0] to [0-2=0]     0, 2, np -> .{ npnl } .: k1gNnPc1
propagate [0-2=0] to [0-2=0]     0, 2, nnpnl -> .{ np } .: k1gNnPc1
propagate [0-2=0] to [0-2=0]     0, 2, nnp -> .{ nnpnl } .: k1gNnPc1
TERM 8: set to
TERM 7: set to k6x?eA
TERM 6: set to k2eAgNnPc1d1
propagate [6-7=6] to [6-7=6]     6, 7, adjp -> .{ ADJ } .: k2eAgNnPc1d1
propagate [6-7=6] to [6-7=6]     6, 7, modifnl -> .{ adjp } .: k2eAgNnPc1d1
propagate [6-7=6] to [6-7=6]     6, 7, modif -> .{ modifnl } .: k2eAgNnPc1d1
propagate [6-7=6] to [6-8=]      6, 8, right_adjpnl -> .{ intr } modif intr .: k2eAgNnPc1d1
propagate [6-8=] to [6-8=]       6, 8, right_adjp -> .{ right_adjpnl } .: k2eAgNnPc1d1
TERM 5: set to
TERM 4: set to k1gInSc4
propagate [4-5=4] to [4-5=4]     4, 5, npnl -> .{ N } .: k1gInSc4
propagate [4-5=4] to [4-5=4]     4, 5, np -> .{ npnl } .: k1gInSc4
4, 9, npnl -> .{ np } ',' right_adjp comma .:
agree_case_number_gender_and_propagate FAILED
TERM 3: set to k2eAgInSc4d1
```

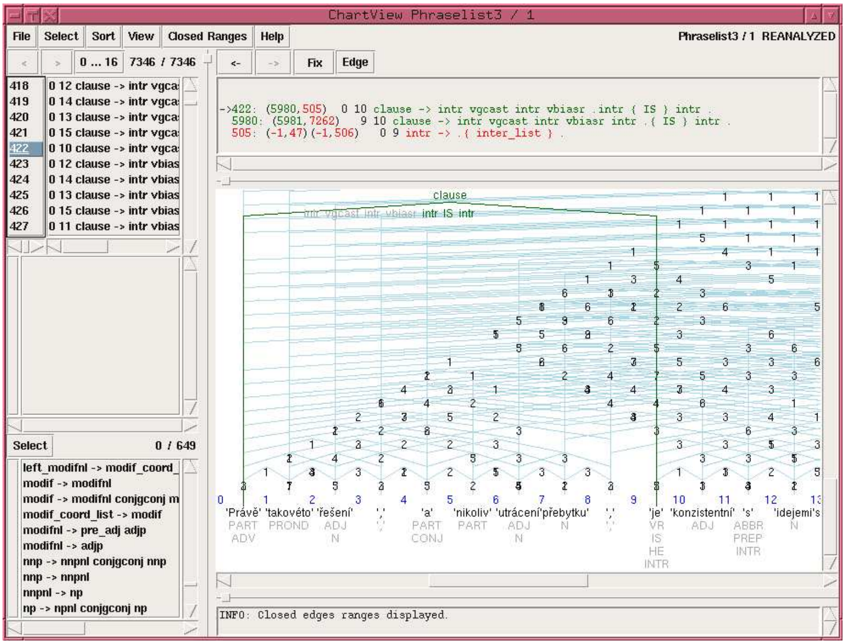
```
INFO: Phraselist corp4000 opened.
```

Figure 3.2: The Gsynt module window showing an analysis of a sentence from corpus.

- ChartView – browser of resulting chart structure.
- GrammarView – grammar forms viewer.
- TBAdmin – tree bank creation tool.

The linguistic work concentrates on the methodology of grammar development with respect to real-world natural language texts. The most important GDW tasks are enhancing the meta-grammar and creation of a tree bank.

The process of building a treebank of correct syntactic trees consists of the following steps: first, each input corpus sentence is analysed in the Gsynt module. The Figure 3.2 shows the basic window with sentences


ChartView Phraselist3 / 1
File
Select
Sort
View
Closed Ranges
Help
Phraselist3 / 1 REANALYZED
0 ... 16
7346 / 7346
<-
->
Fix
Edge
Select
0 / 649
clause
intr vgcast intr vbiasr intr IS intr
'Právě' 'takovéto' 'řešení' ',' 'a' 'nikoliv' 'utrácení' 'přebytku' ',' 'je' 'konzistentní' 's' 'idejemi's
PART ADV
PROND
ADJ N
PART CONJ
PART
ADJ N
N
VR IS HE INTR
ADJ
ABBR PREP INTR
N

```
418  0 12 clause -> intr vgca
419  0 14 clause -> intr vgca
420  0 13 clause -> intr vgca
421  0 15 clause -> intr vgca
422  0 10 clause -> intr vgca
423  0 12 clause -> intr vbias
424  0 14 clause -> intr vbias
425  0 13 clause -> intr vbias
426  0 15 clause -> intr vbias
427  0 11 clause -> intr vbias
```

```
->422:  (5980,505)   0 10 clause -> intr vgcast intr vbiasr .intr { IS } intr .
  5980:  (5981,7262)    9 10 clause -> intr vgcast intr vbiasr intr .{ IS } intr .
  505:  (-1,47)(-1,506)    0 9 intr -> .{ inter_list } .
```

```
left_modifnl -> modif_coord_
modif -> modifnl
modif -> modifnl conjgconj m
modif_coord_list -> modif
modifnl -> pre_adj adjp
modifnl -> adjp
nnp -> nnpnl conjgconj nnp
nnp -> nnpnl
nnpnl -> np
np -> npnl conjgconj np
```

```
INFO: Closed edges ranges displayed.
```

Figure 3.3: The resulting chart in the Grammar Development Workbench tool.

from a corpus.[4] The selected sentence is analyzed by the `synt` parser and the output of the parser is displayed. Several additional information about sentences is presented in the window.

The syntactic trees are displayed by the TreeView module. The resulting syntactic trees can be filtered with several kinds of queries and reduced by successive specification of pruning constraints. The one resulted syntactic tree is added to the tree bank at the end.

Enhancing the meta-grammar lies in determining new rules for an unaccepted sentence. By means of the ChartView module, see the Figure 3.3, a problematic sentence construct that is not covered by the meta-grammar can be located. Then the exact specification of the needed

[4]PDTB-1.0 [Haj98] in this case

rule(s) for the uncovered construct is determined and entered in the GrammarView module, see the Figure 3.4.

We concentrate on the methodology of grammar development with respect to real-world natural language texts. The particular procedures include:

- building a treebank of correct syntactic trees, that is, for a correctly analysed sentence:
  - displaying the syntactic trees. Those trees are all consistent with the imposed meta-grammar and they are ordered by the computed tree ranks. However, the ordering is still premature and the correct tree is often not in the first position yet.
  - reduction of the number of syntactic trees with successive specification of pruning constraints.
  - adding the resulted syntactic tree to the treebank.
- enhancing the meta-grammar, that is determining new rules for an unaccepted sentence:
  - ChartView – helps locating the problematic sentence construct that is not covered by the meta-grammar.
  - GrammarView – for finding the exact specification of the missing rule(s) for the uncovered construct.

For more detailed information about the GDW project see the project documentation page [Vyk07].

3.2 New Meta-grammar Constructs in `synt`

The development of a grammar for syntactic analysis of natural language texts is a tedious process which always tends to keep adding new rules for capturing uncovered language phenomena up to the moment, where the number of rules becomes hardly maintainable. Such “rule overload” is solved with several competing approaches ranging from stochastic parsing [Bod03], through various automatic learning algorithms [CD02] upto loosening the strictness of the analysis with a shallow parsing algorithm [vdBB02].

In the `synt` system, we have decided to apply the meta-grammar concept with three consecutive grammar forms. The underlying analyzer is an augmented head driven parser based on a context free backbone with interleaved evaluation of contextual constraints.

3.2.1 The Meta-grammar Design

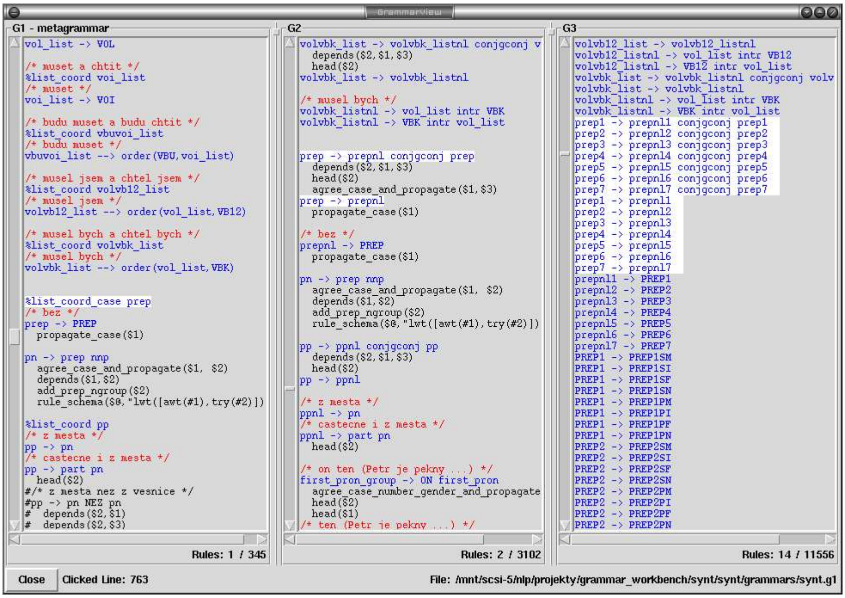
The meta-grammar concept in `synt` consists of three grammar forms denoted as G1, G2 and G3. Human experts work with the meta-grammar form (G1), which encompasses high-level generative constructs that reflect the meta-level natural language phenomena like the word order constraints, and enable to describe the language with a maintainable number of rules. The meta-grammar serves as a base for the second grammar form (G2) which comes into existence by expanding the constructs. This grammar consists of context free rules equipped with feature agreement tests and other contextual actions. The last phase of grammar induction (G3) lies in the transformation of the tests into standard rules of the expanded grammar with the actions remaining to guarantee the contextual requirements.

The Figure 3.4 illustrates the use of the generative construct `%list_coord_case_prep` in G1, that produces two context free rules with pruning actions in G2 and fourteen context free rules in G3. The grammars are displayed by the GrammarView module, which is a part of the Grammar Development Workbench environment described in the previous section.

The number of rules naturally grows in the direction G1 $<$ G2 $<$ G3. The current numbers of rules in the three grammar forms are 236 in G1, 2741 in G2 and 11088 in G3, but the grammar is still being developed and enhanced.

3.2.1.1 The Meta-grammar Form (G1)

The meta-grammar consists of global order constraints that safeguard the succession of given terminals, special flags that impose particular restrictions to given non-terminals and terminals on the right hand side (RHS) and of constructs used to generate combinations of rule elements. Some of these meta-grammar constructs have already been described in [HS00], in this section we present a summary of all the constructs including the latest additions.


GrammarView
Close
Clicked Line: 763
File: /mnt/scsi-5/nlp/projekty/grammar_workbench/synt/synt/grammars/synt.g1

G1 - metagrammar

```
vol_list -> VOL

/* muset a chtit */
%list_coord voi_list
/* muset */
voi_list -> VOI

/* budu muset a budu chtit */
%list_coord vbuvoi_list
/* budu muset */
vbuvoi_list --> order(VBU,voi_list)

/* musel jsem a chtel jsem */
%list_coord volvb12_list
/* musel jsem */
volvb12_list --> order(vol_list,VB12)

/* musel bych a chtel bych */
%list_coord volvbk_list
/* musel bych */
volvbk_list --> order(vol_list,VBK)


%list_coord_case prep
/* bez */
prep -> PREP
  propagate_case($1)

pn -> prep nnp
  agree_case_and_propagate($1, $2)
  depends($1,$2)
  add_prep_ngroup($2)
  rule_schema($@,"lwt([awt(#1),try(#2)])

%list_coord pp
/* z mesta */
pp -> pn
/* castecne i z mesta */
pp -> part pn
  head($2)
#/* z mesta nez z vesnice */
#pp -> pn NEZ pn
#  depends($2,$1)
#  depends($2,$3)
```

Rules: 1 / 345

G2

```
volvbk_list -> volvbk_listnl conjgconj v
  depends($2,$1,$3)
  head($2)
volvbk_list -> volvbk_listnl

/* musel bych */
volvbk_listnl -> vol_list intr VBK
volvbk_listnl -> VBK intr vol_list


prep -> prepnl conjgconj prep
  depends($2,$1,$3)
  head($2)
  agree_case_and_propagate($1,$3)
prep -> prepnl
  propagate_case($1)

/* bez */
prepnl -> PREP
  propagate_case($1)

pn -> prep nnp
  agree_case_and_propagate($1, $2)
  depends($1,$2)
  add_prep_ngroup($2)
  rule_schema($@,"lwt([awt(#1),try(#2)])

pp -> ppnl conjgconj pp
  depends($2,$1,$3)
  head($2)
pp -> ppnl

/* z mesta */
ppnl -> pn
/* castecne i z mesta */
ppnl -> part pn
  head($2)

/* on ten (Petr je pekny ...) */
first_pron_group -> ON first_pron
  agree_case_number_gender_and_propagate
  head($2)
  head($1)
/* ten (Petr je pekny ...) */
```

Rules: 2 / 3102

G3

```
volvb12_list -> volvb12_listnl
volvb12_listnl -> vol_list intr VB12
volvb12_listnl -> VB12 intr vol_list
volvbk_list -> volvbk_listnl conjgconj volv
volvbk_list -> volvbk_listnl
volvbk_listnl -> vol_list intr VBK
volvbk_listnl -> VBK intr vol_list
prep1 -> prepnl1 conjgconj prep1
prep2 -> prepnl2 conjgconj prep2
prep3 -> prepnl3 conjgconj prep3
prep4 -> prepnl4 conjgconj prep4
prep5 -> prepnl5 conjgconj prep5
prep6 -> prepnl6 conjgconj prep6
prep7 -> prepnl7 conjgconj prep7
prep1 -> prepnl1
prep2 -> prepnl2
prep3 -> prepnl3
prep4 -> prepnl4
prep5 -> prepnl5
prep6 -> prepnl6
prep7 -> prepnl7
prepnl1 -> PREP1
prepnl2 -> PREP2
prepnl3 -> PREP3
prepnl4 -> PREP4
prepnl5 -> PREP5
prepnl6 -> PREP6
prepnl7 -> PREP7
PREP1 -> PREP1SM
PREP1 -> PREP1SI
PREP1 -> PREP1SF
PREP1 -> PREP1SN
PREP1 -> PREP1PM
PREP1 -> PREP1PI
PREP1 -> PREP1PF
PREP1 -> PREP1PN
PREP2 -> PREP2SM
PREP2 -> PREP2SI
PREP2 -> PREP2SF
PREP2 -> PREP2SN
PREP2 -> PREP2PM
PREP2 -> PREP2PI
PREP2 -> PREP2PF
PREP2 -> PREP2PN
```

Rules: 14 / 11556

Figure 3.4: Generative construct `%list_coord_case_prep` in the grammar G1 and the appropriate (generated) rules and actions in G2 and G3.

The arrow mark in the rule is used for specification of the *rule type* (`->`, `-->`, `==>` or `===>`). A little hint to the arrow form meaning can be expressed by 'the thicker and longer the arrow the more (complex) actions are to be done in the rule translation'. The smallest arrow (`->`) denotes an ordinary CFG transcription and the thick extra-long arrow (`===>`) inserts possible inter-segments between the RHS constituents, checks the correct order of enclitics and supplies several forms of the rule to make the verb phrase into a full sentence.

The *global constructs* (`%enclitic`, `%order` and `%merge_actions`) represent universal simple regulators, which are used to inhibit some combinations of terminals in rules, or which specify the actions that need some special treatment in the meta-grammar form translation.

The main *combining constructs* in the meta-grammar G1 are `order()`, `rhs()` and `first()`, which are used for generating variants of assortments of given terminals and non-terminals.

```
/* budu se ptat - I will ask */
clause ===> order(VBU,R,VRI)
/* ktery ... - which ... */
relclause ===> first(relprongr) rhs(clause)
```

The `order()` construct generates all possible permutations of its components. The `first()` and `rhs()` constructs are employed to implant content of all the right hand sides of specified non-terminal to the rule. The `rhs(N)` construct inserts all possible rewritings of the non-terminal `N`. The resulting terms are then subject to standard constraints, enclitic checking and inter-segment insertion. In some cases, one needs to force a certain constituent to be the first non-terminal on the RHS. The construct `first(N)` ensures that `N` is firmly tied to the beginning and can neither be preceded by an inter-segment nor any other construct.

There are several generative constructs for defining rule templates to simplify the creation and maintenance of the grammar. One group of such constructs is formed by a set of `%list_*` expressions, which automatically produce new rules for a list of the given non-terminals either simply concatenated or separated by comma and co-ordinative conjunctions.

A significant portion of the grammar is made up by verb group rules (about 40 %). Therefore we have been seeking for an instrument that would catch frequent repetitive constructions in verb groups. The obtained addition is the `%group` keyword illustrated by the following example:

```
%group verbP={
  V:    verb_rule_schema($@,"(#1)")
        groupflag($1,"head"),
  VR R: verb_rule_schema($@,"(#1 #2)")
        groupflag($1,"head"),
}

/* ctu/ptam se - I am reading/I am asking */
clause ====> order(group(verbP), vi_list)
  verb_rule_schema($@,"#2")
  depends(getgroupflag($1,"head"), $2)
```

Here, the group `verbP` denotes two sets of non-terminals with the corresponding actions that are substituted for the expression `group(verbP)` on the RHS of the `clause` non-terminal. In order to be able to refer to verb group members in the rules, where the group is used, any group term can be assigned a *flag* (any string). By that flag an outside action can refer to the term later with the `getgroupflag` construct.

As we can see in the example above, the `verb_rule_schema` action (which defines the part of the verb group that forms a verbal object in the successive logical analysis) appears in both parts of the RHS – the group and the rule itself. For the logical analysis, the arguments of this actions have to be gathered and merged into one resulting action. This behaviour can be defined per-action with the global construct `%merge_actions={verb_rule_schema}`.

Many rules, e.g. those prescribing the structure of a `clause`, share the same rule template — they have the same requirements for intersegments filling and the enclitics order checking as well as the RHS term combinations. To enable a global specification of such majority of rules, we provide a *rule template* mechanism, which defines a pattern of each such rule (the rule type and the RHS encapsulation with some generative construct).

Some grammatical phenomena occur rarely in common texts. The best way to capture this sparseness is to train rule probabilities on a large data bank of derivation trees acquired from corpus sentences. Since preparation of such corpus of adequate size (at least tens of thousands of sentences) is a very expensive and tedious process, we have for now overcome this difficulty with defining *rule levels*. Every rule without level indication is of level 0. The higher the level, the less frequent the appropriate grammatical phenomenon is, according to the guidance of the grammarian. Rules of higher levels can be set on or off according to the chosen level of the whole grammar.

```
3:np -> adj_group
  propagate_case_number_gender($1)
```

In the above example the rule is of level 3, thus when we turn the grammar level to at least 3, we allow adjective groups to form a separate *intersegment*.[5] When analyzing with grammar of level 0 the rule '`np ->`

[5] *intersegment* is a technical name of any constituent that may supplement the verb phrase in a sentence.

adj_group' is not seen as a part of the grammar at all.

Apart from the common generative constructs, the meta-grammar comprises feature tagging actions that specify certain local aspects of the denoted (non-)terminal. One of these actions is the specification of the head-dependent relations in the rule — the `head()` and `depends()` constructs which allow to express the dependency links between rule terms.

3.2.1.2 The Second Grammar Form (G2)

As we have mentioned earlier, several pre-defined grammatical tests and procedures are used in the description of context actions associated with each grammatical rule of the system. The pruning actions include:

- grammatical case test for particular words and noun groups
- agreement test of case in prepositional construction
- agreement test of number and gender for relative pronouns
- agreement test of case, number and gender for noun groups
- type checking of logical constructions

```
np -> adj_group np
  rule_schema($@, "lwtx(awtx(#1) and awtx(#2))")
  rule_schema($@, "lwtx([[awt(#1),#2],x])")
```

The `rule_schema` action presents a prescription for building a logical construction out of the sub-constructions from the RHS. Each time a type checking mechanism is applied and only the type-correct combinations are passed through.

The G2 contextual actions `propagate_all` and `agree_*_and_propagate` compute and propagate all relevant grammatical information from the selected non-terminals on the RHS to the one on the left hand side of the rule.

3.2.1.3 The Expanded Grammar Form (G3)

The feature agreement tests can be transformed into context free rules. For instance in Czech, similar to other Slavic languages, we have 7 grammatical cases (nominative, genitive, dative, accusative, vocative, locative and instrumental), two numbers (singular and plural) and three genders (masculine, feminine and neuter), in which masculine exists in two

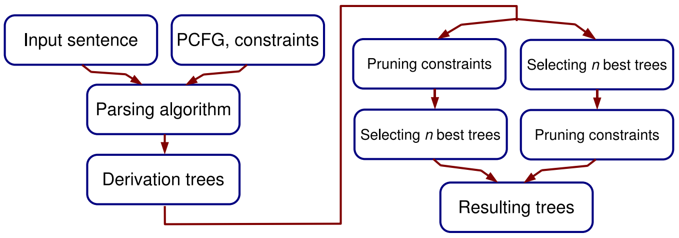


Figure 3.5: Modules and data flow in the `synt`'s parsing librabry `libkp`

forms — animate and inanimate. Thus, e.g., we get 56 possible variants for a full agreement between two constituents.

The G3 grammar represents such transformation of the G2 form with the agreement actions interleaved into the rules. The actual parsing process can work in two modes – either directly with the G2 grammar and separate evaluation of agreement tests or with the G3 grammar, where the agreement tests are inseparable part of the grammar rules. Our experiments [HS00] show that the running times of these two approaches are, in a general case, of nearly the same value – the G3 form performs slightly better for more ambiguous sentences.

3.2.2 The Parsing Algorithm

We restrict our work to lexicalized grammars, where terminals can only appear in lexical rules in the form of $A \rightarrow w_i$.[6] This restriction allows us to simplify the implementation and it also enables to separate a lexicon from the grammar.[7] The parsing module of `synt`, the `libkp` library developed by Vladimír Kadlec in the MU NLP Centre, provides an efficient implementation of standard parser tasks:

- syntactic analysis of sentences in natural language based on context free grammars that can be large and highly ambiguous;
- efficient representation of derivation trees;

[6] i.e. preterminal category → terminal word.

[7] Actually, instead of using a static lexicon, we construct the lexical rules using the Czech morphological analyser `ajka` [Sed05].

- pruning of the trees based on the application of contextual constraints;
- selecting n best trees based on computed edge rank values (e.g. the frequency characteristics obtained from treebanks);
- visualization and printing of the parsing trees in a graphical form.

The parsing process consists of several steps. Firstly, the packed shared forest [Ear70] is produced by the standard context free parsing algorithm. Then the contextual constraints are applied. Finally, the best trees are selected. The order of the last two steps can be reversed. We use this multi-pass approach, because all these functions are implemented as plugins that can be modified as needed or even substituted with other implementations. For example, we have compared four different parsing algorithms which use identical internal data structures (Earley's top-down and bottom-up chart parser [Ear70], head-driven chart parser [Kay89] and Tomita's GLR [Tom86]. All these implementations produce the same structures, thus applying contextual constraints or selecting n best trees can be shared among them. The data flow in the `libkp` library is shown in the Figure 3.5.

The parsing technique of the `synt` system is based on the head driven approach with improvements regarding the process of confirmation of viable hypotheses. The HDddm (head driven with dependent dot move) parsing technique [KS06] refers to the fact that the move of one "dot" in the head driven parsing step is dependent on the opposite move of the other one.

The efficiency of the parser depends to a considerable extent on the choice of grammar rule heads. The current positions of heads in the `synt` meta-grammar have been chosen experimentally and they are in agreement with the conception of the leading constituent in traditional Czech grammars.

3.2.3 Evaluation of Contextual Constraints

The contextual constraints (or actions) defined in the meta-grammar can be divided into four groups:

1. rule-tied actions
2. agreement fulfillment constraints
3. post-processing actions
4. actions based on derivation tree

The rule-based edge ranking is solved on the first level by the rule-tied actions, which also serve as rule parameterization modifiers.

Agreement fulfillment constraints serve as chart pruning actions and they are used in generating the expanded grammar G3. The agreement fulfillment constraints represent the functional constraints, whose processing can be interleaved with that of phrasal constraints.

The post-processing actions are not triggered until the chart is already completed. Actions on this level are used mainly for computation of analysis transformations for a particular input sentence and particular analysis. Some such computations (e.g. verb frame extraction, see the Section 3.4) demand exponential resources for computation over the whole chart structure. This problem is solved by splitting the calculation process into the pruning part (run on the level of post-processing actions) and the reordering part, that is postponed until the actions based on derivation tree.

The actions that do not need to work with the whole chart structure are run after the best or n best derivation trees are selected. These actions are used, for example, for determination of possible verb frames within the input sentence, which can produce a new ordering of the selected trees, or for the logical analysis of the sentence.

The edge rank computations use information from several supporting resources, such as a treebank of testing sentences or a list of verb valencies. The latter one is currently the most promising resource since we are working on the VerbaLex lexicon, see the Section 2.1, featuring syntactic dependencies of sentence constituents, their semantic roles and links to the corresponding Czech WordNet classes.

3.2.4 The `synt` Parser Implementation

In the `synt`, every grammar rule has zero, one or more semantic actions. The actions are computed bottom-up serving the purpose of:

a) computing a value used by another action on the higher level;
b) throwing out incorrect derivation trees.

For example, the following grammar rule for genitive constructions in Czech has three semantic actions:

```
npnl -> np np +0.0784671532846715
  test_gen ( $$ $2 )
```

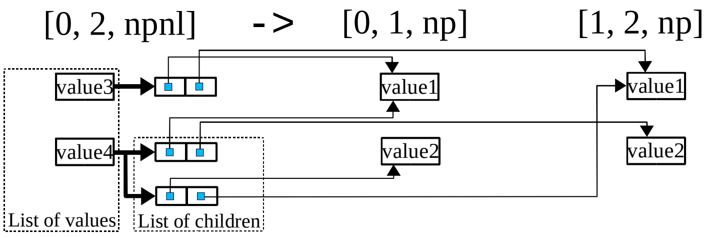


Figure 3.6: Example of the forest of values.

```
prop_all ( $$ $1 )
depends:1 ( $$ $1 $2 )
```

First line contains a grammar rule with its frequency obtained from a treebank. The contextual constraints are listed on the lines bellow it. The number 1 after the colon represents an internal classification of the action. We can turn an evaluation of actions with specified type on and off. The `$$` parameter represents the return value. The $\$n$ parameter is a variable where we store a value of n-th nonterminal of the rule. Notice that the presented notation is not entered directly by users. It is generated automatically from the meta-grammar G1.

3.2.4.1 The Representation of Values

[BBR87] showed that parsing is in general case NP-complete problem if grammars are allowed to have agreement features. The pruning constraints in `synt` are weaker than general feature structures. It allows an efficient implementation with the following properties. A node in the derivation tree has only limited number of values (e.g. the cardinality of the set for noun groups in our system is max. $56 = 7$ cases $\times$ 2 nouns $\times$ $(3 + 1)$ genders [HS00]). In `synt` instead of usual pruning of the original packed share forest, a new forest of values is built during the analysis. The worst case time complexity for one node in the forest of values is therefore 56^{δ}, where δ is the length of the longest right-hand side of grammar rules. Note that this complexity is independent of the input sentence.

Table 3.1: Results of running the `synt` parser on 10 000 corpus sentences (from the DESAM corpus, running on Intel Xeon 2.2GHz).

#sentences	10000 sentences
#words	191034 words
Maximum sentence length	155 words
Minimum sentence length	2 words
Average sentences length	19.1 words
Time of CFG parsing	5 minutes 52 seconds
Time of evaluating constraints/actions	38 minutes 32 seconds
Overall time with freeing memory	46 minutes 9 seconds
Average #words per second	68.97
Size of the log file	1.2 GiB
#accepted sentences	9208

The values in the forest of values are linked with Earley's items.[8] An item contains a single linked list of its values. Each value holds a single linked list of its children. The child is one dimensional array of values. This array represents one combination of values that leads to the parent value. Notice that there can be more combinations of values that lead to the same value. The i-th cell of the array contains a reference to a value from i-th symbol on the RHS of the corresponding grammar rule. In case that the i-th symbol is not used for computation of the parent value, only the reference to Earley's item from such cell is used.

The Figure 3.6 shows an example representing rule `npnl -> np np` and containing three Earley's items. Each item on the RHS of the rule contains two values, *value1* and *value2*. This gives us four possible combinations. The semantic action computes from combinations *value1* × *value2* and *value2* × *value1* the same value *value4*. The combination *value2* × *value2* was classified as incorrect (by the action), so it is not displayed.

During this experiment, `synt` has achieved an average of 92.1 % coverage on the DESAM corpus [PRS97] sentences with about 84 % cases where the correct syntactic tree was present in the result. However, the

[8]For the description of *Earley's items* see e.g. [AH02].

process of automatic determining of the one tree that is correct from the point of view of a linguistic expert is still premature. An average time of the analysis of one sentence from the corpus data was 0.28 seconds.[9] More details of timing results can be found in the Table 3.1.

3.3 Best Analysis Selection – a Supervised Construction of Pruning Constraints

A common approach to acquiring the statistical data for analysis of syntax employs learning the values from a fully tagged tree bank training corpus. Building such corpora is a tedious and expensive work and it requires a team cooperation of linguists and computer scientists. At present the only representative source of Czech tree bank data is the Prague Dependency Treebank (PDT) [Haj04a], which includes dependency analyses of more than 100 000 Czech sentences.

In order to be able to exploit the data from PDT in our non-dependency analysis, we have supplemented the meta-grammar with dependency specification for constituents. Thus the output of the analysis (one selected derivation tree) can be presented in the form of pure dependency tree.

This technique enables us to relate the output of our parser to the PDT data. However, the profit of exploitation of the information from the dependency structures can be higher than that and can run in an automatically controlled environment. For this purpose, we use the mechanism of *pruning constraints*. A set of strict limitations is given to the syntactic analyser, which passes on just the compliant parses. The constraints can be either supplied manually for particular sentence by linguists, or obtained from the transformed dependency tree in PDT.

The transformation is driven by guidelines specified by linguists. These guidelines contain the following information:

- *afun* — analytical function attribute from PDTB 1.0
- *term* — corresponding nonterminal or preterminal from the meta-grammar

[9]The average running time includes generation of log file messages.

#	*afun*	*term*	*mtag*	*lexit*
	Sb	np	k1	
	Sb_Ap	np		
	Obj	np		
	Atr	modif	k2	
	AuxP	pn		

Table 3.2: Simplified example of transformation guidelines.

- *mtag* — morphological tag constraint
- *lexit* — lexical item constraint

The automatic procedure for generating the pruning constraints then successively tries to match the analytical function attribute in the input sentence with the records in the transformation guidelines. Each match found is then checked for agreement in the particular morphological tag and lexical item according to the given criteria (currently a pattern matching based on text regular expressions). If all required fields are consistent with the guidelines, the corresponding subtree is marked as the specified nonterminal or preterminal from the meta-grammar. The syntactic analysis with the pruning constraints applied then prunes those parsing trees from the resulting chart that do not contain the requested nonterminal or preterminal in that position.

If more than one records in the guidelines match, the first match is applied. This mechanism allows to prefer the most specific records to the general ones, which differ in the lexical item constraint or the morphological tag constraint only, used e.g. in the differentiation of various adverbial types:

#	*afun*	*term*	*mtag*	*lexit*
	Adv	np	k1	
	Adv	adv		

The process of transformation guidelines preparation is divided into several steps to assure the consistency of acquired pruning constraints. After

Table 3.3: Examples of the reduction of the number of parsing trees for randomly selected sentences.

sentence #	# words	# analyses	# pruned analyses	reduced to (%)
00214	30	3112504044	2146560	0.07
00217	3	2	2	100
00287	12	56	4	7
00308	7	10	6	60
00486	6	1	1	100
00599	35	44660	4872	11
00612	25	2369063760	1048896	0.04
00842	17	409920	6336	1.5

every change, the results are checked against a testing set of input sentences and the differences are reported to the user for arbitration.

The integration of the pruning constraints obtained automatically with the mechanism of transformation guidelines has shown to be very efficient. The tedious work of the training data acquisition for the best analysis selection algorithm has been substantially facilitated. Examples of the reduction are displayed in the Table 3.3. The Table presents examples of sentences which were randomly chosen from the set of 1000 sentences analysed first without pruning constraints and then with automatically generated pruning constraints.

The average percentage of reduction on all the tested sentences has achieved 30 % (see the Figure 3.7). The future work on the refinement of transformation guidelines will be concentrated on further reduction of the number of automatically pruned analysis trees.

3.4 Parsing with Verb Frame Information

In the case of syntactic analysis of a free word order language, as the Czech language is, we need to exploit the language specific features for obtaining the correct ordering of the resulting syntactic analyses. So far the most advantageous approach is the one based upon valencies of the

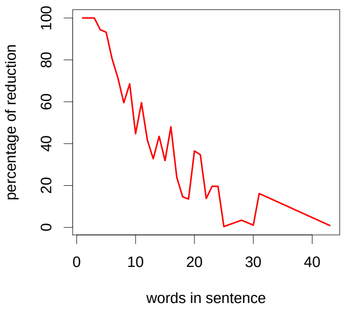


Figure 3.7: The dependency of the reduction (%) of the number of resulting analyses on the number of words in the input sentence

verb phrase – a crucial concept in traditional linguistics, as well.

The ambiguity level in the syntactic analysis of free word order languages suffers from the exponential explosion of the number of resulting derivation trees. The main reasons for this combinatorial grow arise on several levels of the sentence building process (prepositional attachment, verb argument resolution, non-projectivity, ellipsis, morphological ambiguity, etc.). Usual approaches [BN00] solve this situation with probabilistic selection of the "best" tree or trees. The best trees are judged by a probabilistic figure of merit. Our experiments show, that in the case of really free word order languages (like Czech) the probabilistic measures, without a combination with other techniques, are not able to cover the complexity of the sentence syntax. That is why we need to exploit the knowledge of the language specific features as described in [HS02].

The basic sentence frame is driven by the lexical characteristics of its predicative construction based on the set of possible verb valencies of the particular verb (see e.g. [TK98]). We have implemented the technique of discovering the possible verb valencies from the resulting ambiguous packed shared forest (stored in the parsing chart). This enables us to work with verb valencies in two directions: a) using the VerbaLex valency lexicon to prune impossible combination regarding the particular verb, and b) automatically process large corpora for discovering possible verb valencies that are missing in the lexicon. These valencies are then offered to the linguistic expert for addition to VerbaLex. Similar approach has been described in [GAL03], in which a partial parsing outputs were used for obtaining the verb subcategorization information. Our approach includes a full parsing of Czech sentence, which increases the credibility of the verb frame information.

The part of the system dedicated to exploitation of information obtained from a list of verb frames is necessary for solving the prepositional attachment problem in particular. During the analysis of noun groups and prepositional noun groups in the role of verb valencies in a given input sentence one needs to be able to distinguish free adjuncts or modifiers from obligatory valencies. The WordNet classes together with the surface features in complex valency frames are directly used for setting up a set of heuristic rules that determine whether a noun group found in the sentence serves here as a free adjunct or not. The heuristics are based on the lexico-semantic constraints derived from the VerbaLex links to the EuroWordNet hypero-hyponymic hierarchy.

Certainly, when checking the valencies with VerbaLex, we discharge the dependence on the surface order. Before the system confronts the actual verb valencies from the input sentence with the list of valency frames found in the lexicon, all the valency expressions are reordered. By using the standard ordering of participants, the valency frames can be handled as pure sets independent on the current position of verb arguments. However, since VerbaLex contains an information about the *usual* verb position within the frame, we promote the standard ordering with increasing or decreasing the respective chart edge rank.

3.4.1 Automatic Extraction of Verb Frames from the Packed Shared Forest

The verb frame extraction (VFE) process in the `synt` system is controlled by the meta-grammar semantic actions. As we have described in the Section 3.2.4.1, the parser builds a forest of values to represent a result of the application of contextual constraints. The VFE actions are then executed on a different level (see the Section 3.2.1) than the "usual" actions, which allows us to apply VFE actions on the whole forest of values.

First of all, we find all noun groups covered by the particular context free rule. Then compatible groups[10] are processed by the VFE action. Notice, that this step suffers from a possible exponential time complexity because we work with the derivation trees and not with the packed forest. On the other hand our experiments show (see the Table 3.4) that in the average case this is not a problem.

If the analyzed verb has a corresponding entry in VerbaLex, we try to match the extracted frame with frames in the lexicon. An example sentence analysis with the VFE process can be found in the Section 2.1.4.

We have measured the results of the first version of the automatic verb frame extraction on 4117 sentences from the Czech corpus DESAM [PRS97]. We have selected sentences which are analysed on the rule level 0, i.e. sentences, which do not contain analytically difficult phenomena like non-projectivity or adjective noun phrases. Even on those sentences the number of possible valency frames can be quite high (see the Table 3.4). However, if we work with intersections of those possible valency frames, we can get a useful reduction of the number of resulting derivation trees – see the examples described in the next Section.

3.4.2 Examples

The projection of the extracted valency frames to the corresponding VerbaLex entry can be used as an effective pruning tool for decreasing the number of successful derivation trees. As an example of such pruning, we can have a look at the sentence

> Pokud **uchazeči kurs rekvalifikace úspěšně absolvují**, budou mít jistě uplatnění v zaměstnání.
> (lit.) If **the candidates the retraining course successfully complete**, they will certainly assert themselves in their job.

A graph of all closed ranges in the resulting chart of this sentence is displayed in the Figure 3.8. The corresponding valency frame for the verb 'absolvovat' from VerbaLex:

> *absolvovat*:1/*graduate*:1 (receive an academic degree upon completion of one's studies)
> `AG<person:1>`$^{\mathrm{obl}}_{\mathrm{who\_nom}}$ `VERB KNOW<education:1>`$^{\mathrm{obl}}_{\mathrm{what\_acc}}$

[10]Compatible in the term of derivation, i.e. groups within the same derivation tree.

Table 3.4: The results of verb frame extraction from the corpus DESAM.

Number of sentences:	
count	**4117**
Number of words in sentence:	
minimum	2.0
maximum	68.0
average	16.8
median	**15.0**
Number of discovered valency frames:	
minimum	0
maximum	37080
average	380
median	**11**
Elapsed time:	
minimum	0.00 s
maximum	274.98 s
average	6.86 s
median	**0.07 s**

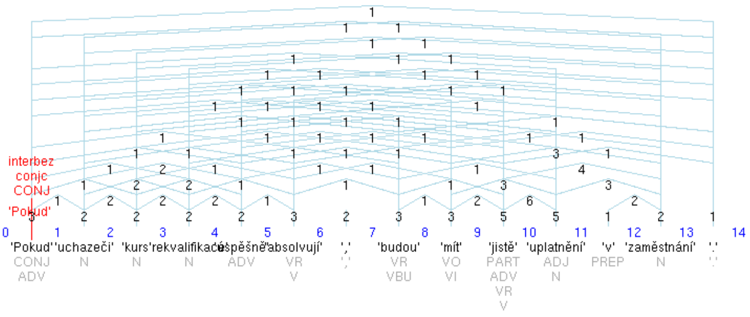

interbez
conjc
CONJ
'Pokud'
'Pokud' 'uchazeči' 'kurs' 'rekvalifikace' 'úspěšně' 'absolvují' ',' 'budou' 'mít' 'jistě' 'uplatnění' 'v' 'zaměstnání' '.'
CONJ ADV N N N ADV VR V VR VBU VO VI PART ADV VR V ADJ N PREP N

Figure 3.8: A graph of all closed ranges in the resulting chart of the sentence “Pokud uchazeči kurs rekvalifikace úspěšně absolvují, budou mít jistě uplatnění v zaměstnání.” (see the Section 3.4.2).

There are 132 trees for that sentence in the parsing system `synt`. Due to the free word order the sequence of sentence constituents is

> *subject* (uchazeči/candidates) – *object* (kurs/course)
> – *verb* (absolvují/complete).

According to the valency frame the subject is a noun in nominative and the object is a noun in accusative. It is evident, that those elements cannot form a nominal phrase. This constriction reduces the number of trees to 24.

Another example is displayed in the following sentence:

> Havel se radil s představiteli justice a vnitra o posílení práva.
> Havel consulted with representatives of judiciary and home office on the consolidation of the legal system.

The valency frame for the verb 'radit se' from VerbaLex is:

> *radit se*:1/*debate*:3 (discuss the pros and cons of an issue)
> `AG<person:1>`$^{\text{obl}}_{\text{who\_nom}}$ `VERB SOC<person:1>`$^{\text{opt}}_{\text{with+whom\_ins}}$
> `ENT|ABS<entity:1|abstraction:1>`$^{\text{opt}}_{\text{about+what\_loc}}$

The number of `synt` trees for this sentence is 2672. The part of sentence with the preposition 's' (with) and a noun in instrumental and the part of sentence with preposition 'o' (on) and a noun in locative are necessarily prepositional nominal phrases. The application of such limits in `synt` allows a significant reduction of the number of trees to 18.

3.5 The Beautified Chart Method – Pruning Technique Based on Linguistic Adequacy

In this section, we present one of the methods used for automatic reduction of the number of output syntactic trees – the method is named the *beautified chart*. We describe the principles of the method as well as its implementation in the `synt` system. We evaluate the impact of the method on the numbers of the resulting derivation trees and particular algorithms on the `synt` internal data structures.

3.5.1 Beautified Trees

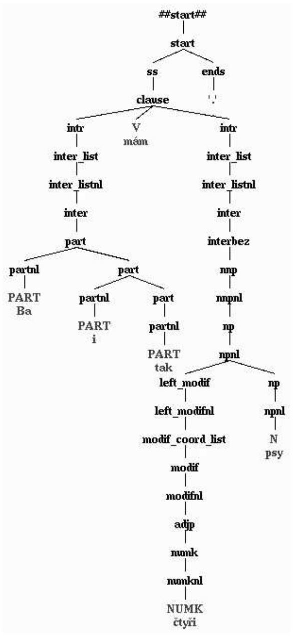
Thanks to the nature of the `synt` meta-grammar concept (for keeping maintainable number of meta-rules), the (generated) grammar of `synt` contains a significant number of non-terminals that have low (or only "technical") impact to the linguistic interpretation of the results. These non-terminals cause the fact that many differences between the resulting derivation trees are also mostly technical. That is why we have introduced a technique that rebuilds the derivation trees in a way, which keeps all the linguistically relevant information and removes only the technical nodes from the derivation trees to make so called *beautified trees.* The difference between a regular tree and a *beautified tree* is shown in the Figure 3.9. As we can see, the beautified trees are also simpler and more comfortable to read than the regular output trees, which is however not the main reason of the method implementation. The main reason lies in the reduction of the number of output trees while keeping all the linguistically adequate features of the trees.

As illustrated in the Figure 3.9, there are 4 basic modifications of the derivation tree:

- *rename* – renames particular non-terminals in the resulting tree (e.g. *ss* → *sentence*). This is a 1:1 operation.
- *ignore* – deletes particular (technical) non-terminals in the resulting tree – the descendants of the deleted node are linked to its parent node (e.g. *intr*, *partnl*)
- *lists* – rebuild the structured non-terminal lists to "flat" form (e.g. the *part* list in the left sub-tree)
- *importance* – selection of one important non-terminal from each non-terminal linear sequence and deleting the others (e.g. the *np* non-terminal was selected from the sequence in the right hand part of the tree in the Figure 3.9)

While transformation of a single tree is not a difficult algorithm, in the `synt` parser, which is based on the head-driven chart parsing algorithm, we need to "upgrade" this method to a transformation of the whole *packed shared forest* of the results to keep a good running time and complexity of the parsing.

A regular derivation tree:



The same tree as a “beautified” tree:

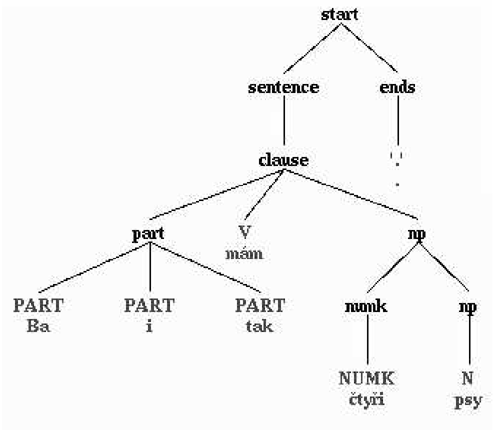


Figure 3.9: Regular tree vs. beautified tree

	#sentences	percentage
Accepted for testing	3393	67.86%
Excluded from testing	877	17.54%
Not accepted	730	14.60%
Total sentences	5 000	100.00%

Table 3.5: Results – all sentences.

The principal idea of the *beautified chart* method consists in an automatic transformation of the `synt` internal data structures (*chart*, *forest of values*) to new structures that will be able to generate (all) *beautified trees* directly. The design of the used transformation operations follows the requirement of *reducing* the structures, so the number of resulting *beautified trees* is supposed to be (significantly) lower than numbers of the original trees.

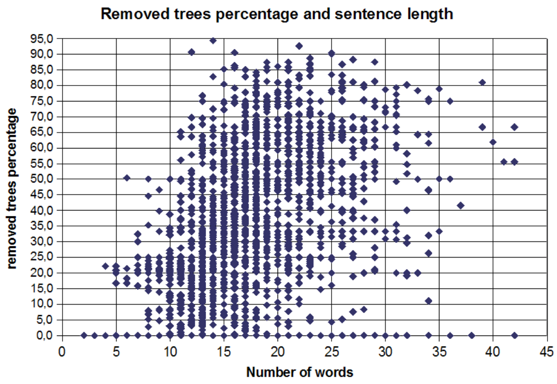
3.5.2 The Previous Estimate of the Effect of the Beautified Chart Method

Before the actual implementation of the presented beautified chart method, we have done a test aimed at finding out how much the (future) method reduces the number of possible output trees. The task was to find out how dramatically the number of derivation trees decreases, if there is some correlation with the sentence length and other input variables.

In the test, we have randomly selected 5 000 Czech sentences of different length (from 2 to 42 words). After processing the sentences with `synt` the number of possible resulting *beautified trees* were computed, which allows to estimate the asset of the method (for this computation we have limited the number of output trees to 50 000). The new statistical quantity *Removed trees percentage* was defined as $100 * (nf - nb)/nf$ where nf = number of full trees, nb = number of *beautified trees.*

According to the parsing results (see the Table 3.5), the sentences were divided into three groups:

- Accepted for testing – the sentence was accepted by the parser, number of regular trees was lower than 50 000 (*Removed trees percentage* was computed only for these sentences).



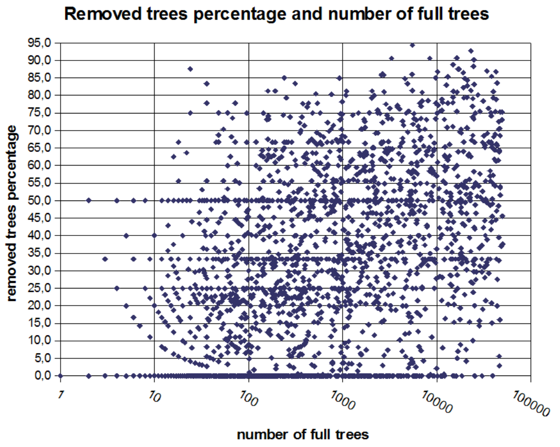


Figure 3.10: Graphical representation of the *Removed trees percentage* characteristics.

Results – short sentences (maximum 12 words):

	#sentences	percentage
Accepted for testing	1351	92.72%
Excluded from testing	0	0,00%
Not accepted	106	7.28%
Total sentences	1457	100.00%

Results – medium-length sentences (13 – 20 words):

	#sentences	percentage
Accepted for testing	1432	84.09%
Excluded from testing	86	5.05%
Not accepted	185	10.86%
Total sentences	1703	100.00%

Results – long sentences (more than 20 words):

	#sentences	percentage
Accepted for testing	610	33.15%
Excluded from testing	791	42.99%
Not accepted	439	23.86%
Total sentences	1840	100.00%

Table 3.6: Results – per sentence length (for short, medium and long sentences).

- Excluded from testing – the sentence was accepted by the parser, number of regular trees was higher than 50 000.
- Not accepted – the sentence was rejected by the parser.

The parser accepted 85 percent of the sample sentences, two thirds of it were acceptable for our measurements. We can see in the Table 3.6 that the parser is (naturally) more successful on short sentences. For sentences with more than 20 words, only 33 percent of sentences were appropriate for our testing. It means that we had only little information about the method's effect on the long sentences. However, the actual

Number of sentences	3393
Average	21.513 %
Maximum Minimum Median	94.309 % 0.000 % 8.333 %
Variance	629.649 %
Standard deviation	25.093 %
Correlation between number of words and removed trees percentage	0.598
Correlation between number of regular trees and removed trees percentage	0.384

Table 3.7: Statistical characteristics of the *Removed trees percentage.*

Number of sentences	2042
Average	32.780 %
Maximum Minimum Median	94.309 % 0.000 % 33.333 %
Variance	629.574 %
Standard deviation	25.349 %

Table 3.8: *Removed trees percentage* - long sentences (more than 12 words).

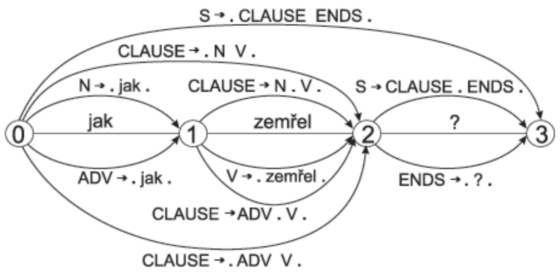


Figure 3.11: Illustration of the *Chart* structure.

implementation showed that the effect of the method on long sentences is even bigger.

Values of the *Removed trees percentage* oscillated between 0 and 94 percent, the average was around 22 percent. High variance and standard deviation indicate that particular results could differ significantly. If only long sentences are taken into consideration, the average and the median were higher (see the Table 3.8). Correlation coefficients and diagrams in the Figure 3.10 also indicated that there is a weak positive dependency especially between the number of all trees and the removed trees percentage. Then, we could expect higher efficiency at sentences with more than 50 000 regular derivation trees. This were the numbers from the pre-implementation estimation of the method's asset.

3.5.3 The Beautified Chart Algorithm

In this section, we describe the internal `synt` structures, *chart* and *forest of values*. We also describe the *beautified chart* transformations of these structures.

Chart (denoted also as the "packed shared forest") is the direct output from the context free stage of the parsing. It is an oriented multigraph where nodes are boundaries between words in the input sentence and edges represent the part of the sentence that has been successfully analysed by a certain grammar rule.

The example of the *chart* structure for an ambiguous Czech sentence "Jak zemřel ?" (ambiguous: 1. How did he die?; 2. Did the yak (animal) die?) is shown in the Figure 3.11.

In addition to this, there are so-called “parent” links between the edges. These links are created in the process of context free analysis and their task is to keep the information about the analysis run in the *chart* structure. Following these links and the grammar rules incident to particular edges, it is possible to generate all derivation trees from the chart.

3.5.3.1 The Beautified Chart Algorithm on the Chart Structure

The *beautified chart* algorithm on the *chart* structure is based on removing particular edges from the structure and redirecting the appropriate “parent” links. The unwanted edges can be detected using the non-terminals on the left side of the appropriate rules (only these non-terminals are used for denominating nodes in the resulting derivation trees). After deleting particular edge, its “parents” are linked to the “child” of the deleted edge.

By implementing the *beautified chart* algorithm on the chart structure we have encountered several complications. The most serious of them was the problem that the chart structure (as the output of context free stage of the parsing process) contains insufficient contextual information. Further, we were unable to perform contextual actions computation on the new “beautified chart” structure because this modified structure is not compatible with the `synt` grammar. Thus, the *beautified chart* algorithm on the chart structure is not compatible with contextual actions evaluation. For this reason, we have decided to implement the *beautified chart* method on the other `synt` internal structure, the *forest of values*.

3.5.3.2 The Beautified Chart Algorithm on the Forest of Values Structure

After finishing the context free part of the parsing process, the contextual actions computation needs to be performed on the resulting *chart* structure. The application of contextual actions (e.g. agreement actions) enables the parser to eliminate high number of incorrect derivation trees. The main output from this process is the *forest of values* structure that contains the “zipped” information of all *successful* derivation trees (i.e.

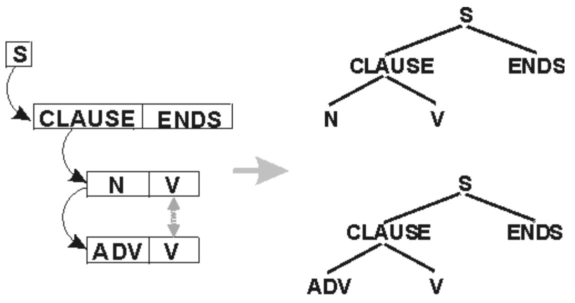


Figure 3.12: Illustration of the *forest of values* structure.

trees which fulfill all the contextual actions).

A simplified *forest of values* structure and the generated derivation trees from this structure are illustrated in the Figure 3.12. The structure's basic unit is an object called *value*. Each value contains a link to an edge from the *chart* structure. In the picture, the values are denoted by the left non-terminal of the rule incident to this edge. Furthermore, each value contains a pointer to a list of "children." Each "child" is an array of pointers to different value objects in the structure. We can see that the length of this array matches to the number of descendants of the appropriate node in the resulting tree, whereas the number of "children" in the list matches to the number of successful sub-trees for the appropriate value. We can also see that there can be more pointers on a single value, which spares memory, but due to this fact, some *beautified chart* modifications are more complicated to implement.

The *beautified chart* algorithm on the forest of values structure is based on replacing particular pointers to the value objects by pointers to their "children" elements. One step of the algorithm is illustrated in the Figure 3.13. As we can see, the pointer to value `v3` is replaced by two pointers, `v5`, `v6` for the first `v3` "child" and `v7`, `v8` for the second one.

The algorithm runs recursively from the bottom (it begins with the values that are attached to the terminal rules). For each value object it computes the list of "beautified children" by searching through its list of children and making modifications like the one shown in the Figure 3.13.

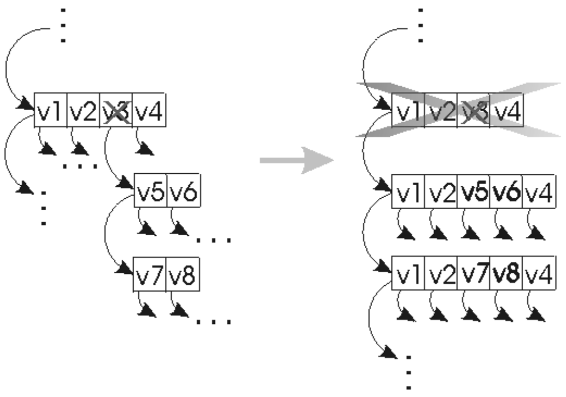


Figure 3.13: The *Beautified chart* modification on the *forest of values* structure

```
compute_b_children (Value *v, Value *old_v)
begin
   rename(v)
   for each w in v->children do begin
      if not computed(w) then
         compute_b_children(w, v)
   end
   while (more_children(v)) do begin
      child = create_next_b_children(v)
      if (sequence(v, child)) then begin
         if (importance(child) > importance(v))
         then begin
            add_b_children(v, child)
            v->for_delete = 2   // removes v
         end
         else
            add_b_children(v, child->b_children)
               // removes child
      end
   end
   if (ignore(v) or list(v, old_v)) then
      v->for_delete = 1    // v will be deleted
   else if (v->for_delete ≠ 2) then
      v->for_delete = 0    // v won't be deleted
end
```

Figure 3.14: The *Beautified chart* algorithm

The basic idea of the algorithm is outlined in the Figure 3.14. The transformations introduced in the Section 3.5.1 are implemented as follows:

- the *rename* transformation is straightforward, it consists in replacement of particular non-terminal names.
- the *ignore* transformation consists in detection of the unwanted values and their replacement as shown in the Figure 3.13. The detection of the unwanted values can be made using the left non-terminals of the grammar rules incident to particular value objects.
- the *lists* modification is similar to the previous one. The only thing which is more difficult is that the value "parent" must be taken into consideration by unwanted value objects detection.
- *importance* is the most complicated transformation. It is based on comparing particular value objects (i.e. the appropriate non-terminals) by their "importance" and deleting the less important ones. In the algorithm, this comparison proceeds before adding the "beautified children" to a particular value. In addition to that, we have to check if both compared value objects are parts of a linear sequence.

During the computation of the "beautified children" lists an additional check must be performed to identify the (rather exceptional) situation where a single (shared) value is deleted from just part of the output trees set and it is kept in the other trees. In such case the value object has to be copied and the lists of children reorganized.

3.5.4 The Beautified Chart Results

The time complexity of the described algorithm is linear with regard to the size of the *forest of values* structure. According to the preliminary results on a small set of Czech sentences, the real effect of the *beautified chart* method is much bigger than we have estimated in the Section 3.5.2. The average values of the *Removed trees percentage* quantity are higher than 50 percent. The total number of trees removed by the *beautified chart* method is even higher, around 90 percent of the total number of regular trees.

Grammar	CT	Atis	PT
Rules	24,456	4,592	15,039
Nonterminals	3,946	192	38
Terminals	1,032	357	47
Test sentences	162	98	30
Average Parses	5.4	940	more than 2^{64}
Grammar	CT	Atis	PT

Table 3.9: Test grammars and test sentences.

We explain these high numbers by the fact that the effect of the described method is much higher in case of sentences that were excluded from testing in the Section 3.5.2. Our measurements show that the tree number reduction is much higher for these sentences.

3.6 Parser Comparison Experiments

The effectivity comparison of different parsers and parsing techniques brings a strong impulse to improving the actual implementations. We have published two experiments [HKS02, HHKK07] comparing the `synt` parser to Moore's parser of English and a set of Czech parsers.

3.6.1 `synt` and Moore's parser

In the first comparison, we have measured the running times of the `synt` parser when analyzing the data provided at the Parser Comparison Data web pages [MC07]. These web pages resulted from discussions at the Efficiency in Large Scale Parsing Systems Workshop at COLING 2000, where one of the main conclusions was the need for a bank of data for standardization of parser benchmarking.

The best results reported on standard data sets (ATIS, CT and PT grammars) before this experiment, were the comparison data by Robert C. Moore [Moo00a]. In the package, only the testing grammars with input sentences are at the disposal.

The basic characteristics of the testing grammars are presented in the Table 3.9. A detailed description of these grammars is given in [Moo00b].

ATIS grammar, Moore's LC_3 + UTF	11.60
ATIS grammar, `synt`	4.19
CT grammar, Moore's LC_3 + UTF	3.10
CT grammar, `synt`	4.19
PT grammar, Moore's LC_3 + UTF	41.80
PT grammar, `synt`	17.75

Table 3.10: Running times comparison (in seconds)

The results of the parser comparison appear in the Table 3.10. The values in the Table give the total CPU times in seconds required by the parser to completely process the test set associated with the grammar.

Since we could not run the referential implementation of the respective Moore's parser on the same machine, the above mentioned times are not fully comparable (we assume that our tests were run on a slightly faster machine than that of Moore's tests). The long running times for the data of the CT grammar are caused by little ambiguity of the grammar, so that the `synt` parsing technique, which is optimized for highly ambiguous grammars, cannot display its strong suits.

3.6.2 Phrasal synt Compared with Dependency Parsers

During last years a number of syntax parsers of the Czech language have been implemented with the concentration to parsing of real texts (in contrast to theoretical and demonstration parsers created in 80s and 90s of the last century).

Some of those "real text parsers" came into existence in the team around the Prague Dependency Treebank [Haj04a], we will call them the Prague parsers although the best ones of them are variants of parsers of British or American authors.[11]

The other set of compared parsers are variants of the parser designed and implemented in the team of the NLP Centre at Masaryk university in Brno (the `synt` parser), thus we call it the Brno parser in the context of this comparison.

[11]see bellow for citations

Although all these parsers have been tested and used for several years already, their implementations are running more or less independently and no rigorous comparison of their effectiveness has been done yet.

Here we also try to formulate the main problems that have hindered such comparison so far, then offer a solution of them and finally present the results of the actual comparison. The Prague parsers have already been compared and rated all together, so the novelty in this comparison is the Brno parser `synt` that is based on completely different approaches in contrast to the Prague parsers.

3.6.2.1 The Prague Parsers – Basic Characteristics

The set of dependency parsers selected and denoted as the Prague parsers contains the following representatives:

McD McDonnald's maximum spanning tree parser [McD06],

COL Collins's parser adapted for PDT [HCRT99],

ZZ Žabokrtský's rule-based dependency parser [HŽ06],

AN Holan's parser ANALOG – it has no training phase and in the parsing phase it searches in the training data for the most similar local tree configuration [Hol05],

L2R, R2L, L2R3, R2L3 Holan's pushdown parsers [Hol04],

CP Holan's and Žabokrtský 's combining parser [HŽ06],

The selection of Prague parsers was limited to the parsers contained in CP, which is currently the parser with the best known results on PDT including also other parsers like Hall and Novák's corrective modeling parser [HN05] or Nilsson, Nivre and Hall's graph transformation parser [NNH06]. These parsers were not included in the comparison, since currently we do not have their results for all sentences of the testing data set.

The Holan's pushdown parsers, during their training phase, create a set of premise-action rules, and apply it during the parsing phase. In the training phase, the parser determines the sequence of actions which leads to the correct tree for each sentence (in case of ambiguity, a pre-specified preference ordering of the actions is used). During the parsing phase, in each situation the parser chooses the premise-action pair with the highest score. In the tests, we have measured four versions of the pushdown parser: L2R – the basic pushdown parser (left to right), R2L – the parser processing the sentences in reverse order, L23 and R23 – the parsers using 3-letter suffixes of the word forms instead of the morphological tags.

3.6.2.2 The Principal Differences of the Parsers

In contrast to the Prague parsers, the Brno parser `synt` is based only on its meta-grammar, the parser does not have any training phase used to learn the context dependencies of the input texts. All rules that guide the analysis process are developed by linguistic and IT experts with all the drawbacks it can bring (see the Section 3.6.2.2 for a description of some of them).

The most principal difference between the two sets of parsers is, of course, the underlying formalism and methodology of the parsing process. This is however not the sort of difference that would cause problems in the parser comparison. In this section, we will concentrate on the problems arising with different input and output data structures, different morphological and syntactic tagging and different presuppositions on the input text that all need to be resolved before we can start with the real comparison.

Q1: The Input Format The input of the Brno parser is either a tagged text (from corpus or from other tagged source) with morphological tags compatible with the tagset of the Czech morphological analyser `ajka` [Sed05] or a plain text (divided into sentences), which is then processed with `ajka`. Since `ajka` does not resolve ambiguities on the morphological level,[12] the Brno parser generally counts with the possibility of ambiguous surface level tokens.

The Prague parsers use as their input also text split into individual sentences, but with unambiguous morphological tags obtained from Hajič's morphological analyser and tagger [Haj04b].

Both morphological analysers (and thus both parser groups) use different morphological tagging systems, which are not 1:1 translatable to each other. However, the differences do not affect the most important

[12]`ajka` provides all possible combinations of morphological features of the input words.

morphological features from the point of view of the syntactic analysis, so we have used an automatic conversion with some information stripping.

Q2: Dependency Trees vs. Phrasal Trees The output of Prague parsers is formed by dependency trees or graphs, whereas the output of the Brno parser is basically formed by packed shared forest of phrasal trees. The Brno parser includes the possibility of sorting the trees of the shared forest and output n trees with the highest *tree rank* (a value obtained as a combination of several “figures of merit,” see [HS02]).

This difference in the output format plus the fact that the Brno team does not yet have a large testing tree bank of phrasal trees for measurements[13] was the cause of the biggest problems in the comparison. Since the measurements had to be done on several thousands of sentences, we have decided to use the PDT-2.0 tree bank[14] [Haj04a]. Since this tree bank provides only the dependency trees,[15] we have decided to convert them to phrasal trees using the Collin’s conversion tool [Col98] and then measure the differences between the Brno parser output and this “phrasal PDT-2.0” using the *PARSEVAL* and the *Leaf-ancestor assessment* techniques (for more details see the Section 3.6.2.3).

Q3: One Resulting Tree vs. (Shared) Forest The output of the Brno parser is formed by the resulting *chart* structure, which encompasses a whole forest of derivation trees (all of them, however, have the same root nonterminal that represents the successful analysis).

In order to be able to provide a comparison of this forest with the one tree obtained from PDT 2.0 conversion procedure, we have for each sentence extracted first 100 (or less) trees sorted according to the *tree rank*. Each of these trees was then compared to the one from PDT and the results are displayed with the following 3 numbers: a) *best trees* – one tree from the set that is most similar to the desired tree is selected and compared; b) *first tree* – the tree with the highest tree rank is selected and compared; and c) *average* – the average of all trees is presented.

[13]Such tree bank of about 5 thousand phrasal trees is being prepared during this year.

[14]The Prague Dependency Treebank, version 2.0, was created by the Institute of Formal and Applied Linguistics, `http://ufal.mff.cuni.cz`.

[15]for more than 80 000 Czech sentences

Q4: Projective vs. Non-projective Trees The output of the Brno parser is always in the form of projective trees, but a non-projective phrase can, in some cases, be analysed with the mechanism of different rule levels, that allow to handle special kinds of phrases. Nevertheless, the Brno parser is not suitable for full analysis of non-projective sentences at the moment. In the future, we will have to provide techniques like corrections for non-projective parses described in [HN05].

On the other hand, the output of the Prague parsers, as a set of dependency edges between words, can cross the word surface order without problems. Thus it can represent projective as well as non-projective sentences.

According to the Prague Dependency Treebank statistics, PDT contains approximately 20 % of non-projective sentences. The sentences selected for comparison are thus not limited to only projective sentences, but the results are counted separately for projective and non-projective sentences.

Q5: The Testing Data Set For the measuring and comparison of parser effectiveness, we definitely need a set of data with syntactic annotation. Such data are available for the dependency trees in PDT. The tree bank has three parts – the training part (train), the testing part for development (d-test) and the testing part for evaluation (e-test).

Since the Prague parsers use the first two sets for development and because there is no such similar tree bank available for the phrasal trees from the Brno parser, we have decided to use the PDT e-test part (approximately 10 thousand sentences) for the comparison and we try to overcome the differences between the parser outputs.

One important difference regarding the testing data set is the fact that the Brno parser does not have any training or learning phase – it is purely grammar-rule based parser. The drawback of this fact is that the Brno parser cannot automatically adapt to kinds of texts that were not intended for analysis. The parser is designed to analyse only sentences of the usual structure. Since the Czech language is a representative of free word order languages, the parser allows an analysis of many possible word combinations that can form even very "wild" Czech sentences, however, it refuses to analyse texts like PDT sentences e-test#00017: "*10 - 3 %*" or e-test#00554: "*Dítě 4 - 10 let : 1640* (Child 4–10 years:1640)." The Prague

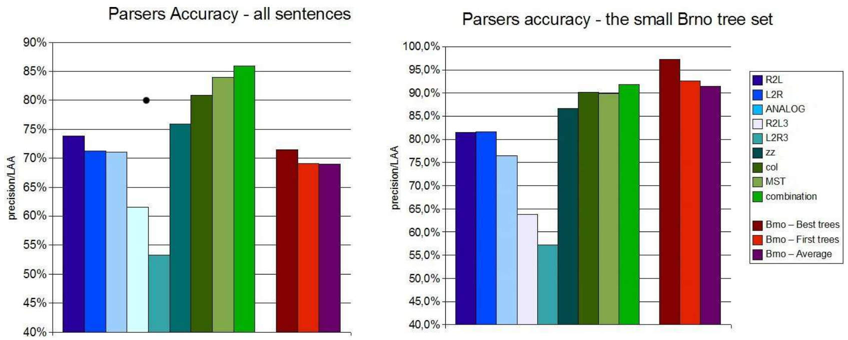


Figure 3.15: The difference of the results with measuring on the converted PDT trees and on the small Brno tree set

parsers, thanks to their stochastic nature, do not have any problems in analysing such kinds of "sentences."

3.6.2.3 The Results

We have used the PDT-2.0 e-test part, where the morphological tags were automatically converted from the Prague tags to the `ajka` tags without ambiguities. The e-test set contains approximately 10 000 syntactically annotated dependency trees. To get trees comparable to Brno parser output, we needed to convert these dependency trees to phrasal trees.

The conversion proceeded in two steps: first, the PDT-2.0 dependency trees in PML format (the default format in PDT-2.0) were converted into the CSTS format (earlier format of PDT) with PDT tool `btred`. Then, the Collin's conversion tool [Col98] was used to obtain PDT-2.0 phrasal trees similar to the output of the Brno parser. The statistical features of the e-test set are:

- 10148 sentences (173586 words)
- 7732 projective sentences
- 2416 non-projective sentences
- 87.7 % Brno parser coverage

Since the Brno parser does not provide output for all sentences in the e-test set (see the discussion in the Section 3.6.2.2), the actual comparison

Table 3.11: The results of the Prague parsers (precision = recall)

Parser	all sentences	non-projective	projective
R2L	73.845 %	69.823 %	75.735 %
L2R	71.315 %	67.297 %	73.204 %
ANALOG	71.077 %	66.625 %	73.169 %
R2L3	61.648 %	58.276 %	63.233 %
L2R3	53.276 %	49.672 %	64.912 %
zz	75.931 %	74.177 %	76.755 %
col	80.905 %	75.634 %	83.383 %
MST	83.984 %	82.230 %	84.809 %
CP	85.85 %	83.434 %	86.979 %

was run only on those sentences from e-test, that were accepted by the Brno parser.

3.6.2.4 Measuring Techniques

The methodology for measuring the results of dependency parsing is usually defined as computation of the precision and recall of the particular dependency edges in the resulting graph/tree. These quantities are measured for each lexical item and the result is then computed as an average precision and average recall throughout the whole set.

In the case of phrasal trees we use the two following measures, *PARSEVAL* and *Leaf-ancestor assessment* (LAA).

The PARSEVAL scheme utilizes only the bracketing information from the parser output to compute three values:

- *crossing bracket* – the number of brackets in the tested analyzer's parse that cross the tree bank parse.
- *recall* – a ratio of the number of correct brackets in the analyzer's parse to the total number of brackets in the tree bank parse.
- *precision*– a ratio of the number of correct brackets in the analyzer's parse to the total number of brackets in the parse.

Table 3.12: The results of the Brno parser on the e-test set

	cross-brackets	precision	recall	LAA
all sentences				
Best trees	4.473	60.228 %	60.645 %	71.5 %
First trees	6.229	47.306 %	50.778 %	69.1 %
Average	5.799	45.627 %	46.584 %	69.0 %
projective sentences				
Best trees	3.619	66.718 %	68.663 %	73.1 %
First trees	5.289	53.028 %	57.630 %	70.6 %
Average	4.942	50.859 %	52.552 %	70.5 %
non-projective sentences				
Best trees	7.251	39.615 %	35.727 %	65.6 %
First trees	9.325	29.275 %	29.699 %	63.5 %
Average	8.625	29.112 %	28.097 %	63.3 %

Table 3.13: The results of the Brno and Prague parsers on the small Brno tree set

	cross-brackets	precision	recall	LAA
Best trees	0.792	89.519 %	92.274 %	97.2 %
First trees	2.132	70.849 %	74.358 %	92.6 %
Average	2.311	63.330 %	64.453 %	91.4 %
R2L		81.472 %		
L2R		81.634 %		
ANALOG		76.537 %		
R2L3		63.754 %		
L2R3		57.201 %		
zz		86.650 %		
col		90.129 %		
MST		89.889 %		
CP		91.912 %		

There are several known limitations [BSDH98] of the PARSEVAL technique. It is not clear whether this metric can be used for comparing parsers with different degrees of structural fineness since the score on this metric is tightly related to the degree of the structural detail.

Also, in case of this comparison there is a problem with evaluating PARSEVAL on non-projective sentences. Processing non-projective sentence, the Collins' conversion tool changes the word order in the resulting phrasal tree, so the PARSEVAL numbers are expected to be bad.

The Leaf-ancestor assessment [Sam00, SB03] measure is more complicated than PARSEVAL. It considers a lineage for each word in the sentence, that is, the sequence of node-labels found on the path between leaf and root nodes in the respective trees. The lineages are compared by their edit distance, each of them having the score between 0 and 1. The score of the whole sentence is then defined as the mean similarity of the lineage-pairs for its respective leaves.

Since it considers not only boundaries between the phrases, the LAA measure is supposed to be more objective than the PARSEVAL, even at non-projective sentences. In this comparison we used the Geoffrey Sampson's LAA implementation [Sam07].

3.6.2.5 Problems and Discussion

Overall results of the Prague parsers testing are presented in the Table 3.11 in the form of percentage of correct dependencies for the whole set of sentences, for non-projective and for projective only. The results of the Brno parser on the whole testing set (with manual tagging from PDT-2.0), e-test is displayed in the Table 3.12.

The experiment of comparing the results of parsers with dependency and phrasal outputs has opened several problems that we have tried to cope with. One of the main causes of these problems were the incompatibilities between the "phrasal PDT" trees and phrasal trees from the Brno parser. This was also the main source of low precision and recall of the parser. In order to prove this thesis, we have (manually) prepared a small set of phrasal trees[16] in the form of the Brno parser trees and repeated the measurements for this subset. The improvement of the results of the Brno parser on this small subset may be seen in the Table 3.13 and in the Figure 3.15.

[16]for 100 sentences randomly chosen from the e-test projective sentences

3.7 Further Development of `synt`

In this chapter, we have described several tools, methods and systems that together form a platform for full syntactic analysis of free word order natural language, the Czech language. The development of the platform components is definitely not yet completed, however, already at this stage the system offers very high coverage on common corpus sentences while keeping the analysis time on the level suitable for batch processing of large texts.

The methods of the best analysis selection algorithm show that the parsing of inflectional languages calls for sensitive approaches to the evaluation of the appropriate figures of merit. The acquisition of these output arranging quantities is based on a representative training data set. The method of pruning constraints described in this text enables to automate the process of treebank corpus transformation.

The integration of the presented methods to the parsing system has no destructive impact on the efficiency of the parser. This is documented by the comparison of the running times. In the described experiment with Moore's parsers, our system outperforms the results of the best referential parsing system on highly ambiguous grammars, for which it is optimized. In the second experiment, we have described a thorough comparison of the techniques and outputs of the two groups of parsers of the Czech language – the stochastic dependency Prague parsers and the meta-grammar phrasal Brno parser. We have summarized and discussed the problems of a comparison of such different approaches and we have presented the measured results of the experiment. The results show that the Prague stochastic parser are better for general textual data, which do not have to follow (Czech) grammatical structures. However, it is not easy to give such conclusion for proper sentences.

In the future development, we would like to repeat these tests on another set of input data, namely on the prepared Brno phrasal tree bank. The question is whether this different testing set will shuffle the table of results significantly or it will stay more or less the same.

The presented parsing system `synt` has already proven its abilities in analysis of running texts in both speed and coverage of various sentence types. With continuous development of the grammar in order to obtain coverage of all the necessary language phenomena, we obtain a quality general purpose system for deep syntactic analysis of natural language texts even for language with such extent of non-analytical features as the Czech language is.

The current development of the system lies mainly in ordering of the obtained analyzes with the usage of language specific features such as the complex valency frames in the VerbaLex valency lexicon. The preliminary results of the exploitation of VerbaLex in the syntactic analysis of Czech are very promising and the precision of the analysis grows significantly. We believe that with enlarging the lexicon to a representative number of Czech verbs the `synt` system will be able to detect the correct derivation tree in many cases which were unsolvable so far.

In the future development, we want to explore and implement methods aimed at reducing the number of output derivation trees and the best analysis selection. Systematic dealing with the syntax ambiguity of the Czech language is the main task for future development of the `synt` parsing system.

Chapter 4

Transparent Intensional Logic as a Way to Semantics

Knowledge representation and reasoning systems usually aim at applications where the speed of responses to questions force using an underlying formalism with limited expressive power. With this approach it is feasible to describe a selected field of interest with a possibly huge amount of facts and reuse this “knowledge” in an expert system tool.

Another approach, with a very small number of real implementations, is based on the fact that most of the human knowledge is encoded in the form of natural (non-artificial) language. Thus a straightforward way to handle such information is to build a system capable of analyzing the sentences directly with a machine parseable output and a connection to an automatic inference machine.

A system that will fulfil such requirements is based on Tichý’s Transparent Intensional Logic (TIL, [Tic88]). The work [Hor02] describes the logical system based on the Normal Translation Algorithm (NTA) for TIL. It follows the approach of full natural language analysis in the form of *normalized TIL constructions* (denoted as *concepts*) as a meaning bearer for natural language expressions and assertions.

In this chapter, the types of lexicons necessary for the TIL logical analysis will be described. We will show the algorithm for analysing the TIL verbal object as the core of a clause construction within the sentence analysis. Examples of verb frame analysis for Czech words will be presented. We also depict a way of enriching the lexicon entries as combinations of the descriptions of lexical units as they are developed within the area of lexical semantics (e.g. WordNet) with logical analysis of sentence meanings worked out within the Transparent Intensional Logic framework.

In the following text, we propose a method to integrate the logical analysis of sentences with the linguistic approach to semantics, exploiting the complex valency frames (CVFs) in the VerbaLex verb valency lexicon presented in the Section 2.1. Since so far these two ways of description, namely the logical and linguistic one, have been treated separately, the task we set is to propose a method of their interrelation and coordination. Needless to say that both ways of description of verb semantics are useful. Hence we are going to show how to combine a logical description using mostly terms like types, individuals, classes, relations, propositions, or, in general, constructions of these entities, with the linguistic framework capturing the idiosyncratic semantic features of verbs. In the Section 4.3 we adduce examples of the analysis of selected English and Czech verbs for which the above mentioned integration has been proposed.

A question may be asked why we do not exploit first order predicate logic (PL1) where some of the presented problems have already been explored with PL1 used to represent logical forms. Nevertheless, it is a well established fact that PL1 is not able to handle systematically the phenomena like intensionality, belief attitudes, grammatical tenses and modalities (modal verbs and modal particles in natural language). On the other hand, since TIL works with techniques designed for capturing the natural language meaning these problems either do not arise or they can be solved in an intuitive way in TIL (see [Tic88]).

The following text further extends the work described in [Hor02, chapters 5 and 6] and summarizes the results published in [Hor04, HP04, HPDM07, GH07a].

4.1 Overview of the Transparent Intensional Logic

The details of the design of TIL have been described in several books, see e.g. [Tic88, Tic04, Mat04, DJM08, Hor02]. Here we will present just a very short overview.

TIL has come out of the ideas of Gottlob Frege, mostly following his fundamental predicament about meaning, known as the Compositionality Principle: the meaning of a compound is a *function* of the meanings of its constituents.

One of the main assets, TIL brings to natural language analysis, is its (hyper)intensionalistic feature, which is enabled through the possibility of *using* and *mentioning* objects of complex types, i.e. not only objects of the universe (individuals), but also functions of individuals, functions of functions of individuals, etc.

The mechanism of intensions and extensions can be displayed with the following example of two expressions: *the dean of the Faculty of Informatics* and *Jiří Zlatuška*. The latter expression is a label of an individual, which is independent on the state of the world and on the time moment. In the case of the first expression the situation is different. The dependency of this expression on time moment is clear: the dean of the Faculty of Informatics was not the same in every time moment of the history. It was, for example, also *Luděk Matyska* or *nobody* before 1994 when the faculty was established. The fact that *Jiří Zlatuška* is now the dean of the Faculty of Informatics is only one of many possibilities, that could have happened. We thus analyse this fact with an intension – a function returning a truth value according to a selected possible world and time moment. In TIL, objects depending on the state of the world and on time moment are called *intensions*, other objects are *extensions*. Non-intensionalistic logic systems, like PL1 which is often used to represent real world facts, have no means to work with intensions.

Another powerful feature of TIL is the extended type hierarchy – TIL works with objects of higher order type. That means we can represent belief sentences in the same way as any other object and that the representations do not lead to any inconsistencies in further processing. A good exemplification is a statement about numerical calculation: if we say *Paul counts the sinus of* $\pi/2$, a first order type representation would make the expression *the sinus of* $^{\pi}/_{2}$ indistinguishable from the number 1. However, this obviously does not reflect the real meaning of the sentence – Paul does not really need to know what is the correct result of *the sinus of* $^{\pi}/_{2}$ and thus he may not even think about *any* particular number here. The TIL analysis of this sentences solves this by relating Paul to the *construction* of the respective mathematical object.

4.1.1 TIL Types

The type system of TIL was based on the theory of types introduced by Church and was then further extended by Tichý [Tic82]. In TIL, every object (as a representative of an entity described by the analyzed natural language expression) has its *type* which is defined over a firmly set type base.

Every TIL object is assigned either one of the *basic types*, or a type that is formed by a *mapping* from one type to another type, or a *higher-order type*. Within this framework, we can obtain an infinitely nested hierarchy of types, i.e. mappings of basic types, mappings of mappings, mappings of mappings of mappings, etc. Nevertheless, how difficult so-ever the mapping is, the object of the respective type is still "flat." The flatness of mappings is predicated upon the way mappings are treated – as collections of $(n+1)$-tuples (in case of an n-adic mapping). This means that the mapping is (virtually) represented by a table of values without any possibility to find out the way (a procedure) which leads to those values. This is also one of the reasons why mere mappings cannot serve as surrogates for meaning – mappings lack a constructing *structure*.

For capturing this structural property, we do not work directly with TIL objects. We rather reference them by the schema describing, how (in correspondence with the natural language expression) the TIL object is *constructed*.

A construction is usually displayed in the typed λ-calculus notation, as a λ-term, but the real construction is still the procedure described with the term, not the term itself.

Variables are the only simple constructions. There are three modes of forming constructions (from non-constructions and other constructions): *trivialization*, *composition* and *closure*. Classes of all constructions form another basic types in the hierarchy.

The idea of logical analysis of natural language with TIL lies in the presupposition that every language has a definite *intensional base* – a collection of fundamental properties of objects (not only individual properties of type $(o\iota)_{\tau\omega}$ but any world and time depending relations among objects like colors, heights, attitudes, ...), that are capable (without any need for other extra techniques) of describing a (thinkable) state of the world.

In TIL, such an intensional base of a natural language is rigorously explicated in an *epistemic framework*, i.e. a typed system based on a set of four basic types of order 1 (simple types) and collections of all constructions of order n as types of order $n + 1$ (higher-order type). These higher-order objects allow us to work not only with extensions and intensions as representatives of real-world objects, but also with so called *hyperintensions*,[1] i.e. the constructions of intensions in the role of TIL objects.

The four simple types form a *type base* and are denoted with letters o, ι, τ and ω, with the following interpretations:

o is a set of two items representing the *truth-values* True (T) and False (F). These two objects behave exactly the same as their counterparts in the standard predicate logic especially in combination with standard logic operations such as conjunction, disjunction, implication or negation. These predicate logic operations can be represented as objects of type (oo) or (ooo), i.e. functions with one or two arguments of type o returning a value of type o.

ι is a class of *individuals*. The designation "individual" must not entice us to imagine the members of this class as beings with all their properties. In TIL, the notion of an individual is best interpreted as a mean of a numerical identification of a (type unstructured[2]) entity. Any individual properties are ascribed to an object of type ι only by means of asserting a statement that contains the ascribing as its part – in the proposition 'The Earth is flat,' we use the ι--object Earth as an identification of an entity that is ascribed the property being flat (Flat)

$$\lambda w \lambda t [\mathsf{\textbf{Flat}}_{wt} \mathsf{\textbf{Earth}}]$$

[1]The term "hyperintensional" has been introduced by Max Cresswell in [Cre75]. See also [Cre85].

[2]i.e. whose type is not decomposable as a function to the types of the functional arguments and the type of its return value.

The individual Earth itself is carrying no *a priori* properties, it serves as an identifier of a further unspecified object and is mainly used for references to this object.

τ is a class of *time moments.* Due to the continuity feature of this class, it may be regarded identical with the class of real numbers in case we specify a fixed zero point and a unit. Functions working with arguments of type τ are usually used for expressing the *temporal dependency* of an entity.

ω is a class of *possible worlds.* Its members are intended for a transparent representation of *modal dependency* of described objects.

A possible world, in accordance with the Leibnizian claims, is in TIL defined as a *determination system* that, for each of the intuitively, pre-theoretically given features from the intensional base, contains an assignment of all possible (consistent) distributions of those features.

Thus the selection (anchoring) of one such possible world allows us to work with values that are dependent on the *actual world* without the need of knowing exactly what the actual world is. This is necessary, since the knowledge of the actual world itself would bring severe inconsistencies into any theory that would rely upon it.

The non-basic types are classes of special entities build upon this base. They form an infinite hierarchy. The classes are formed by mappings from and into the initial objects, mappings from and into those mappings, etc. For every type a countable set is given, whose members are called *variables.*

The types can be also viewed as classes of TIL objects, i.e., if ξ is a type, $\mathsf{A} \in \xi$ then A is called a ξ-object. In TIL, we often work with objects of particular types. The most frequent derived types are described in the Table 4.1.

4.2 The Logical Analysis of a Sentence

For the purpose of obtaining the logical analysis of a sentence within the Normal Translation Algorithm (NTA, [Hor02]), we describe the proce-

Table 4.1: The most frequent derived types in TIL (ξ in the table stands for any type). Also the short notations are displayed.

Type	**Object**
$((o\tau)\omega)$, $o_{\tau\omega}$, π	a *proposition* – an assertion whose truth-value depends on the world and time. **Example:** ‘John is old.’
$((\iota\tau)\omega)$, $\iota_{\tau\omega}$	an *intensional role* – an individual office that may be engaged by different individuals in different worlds or times. **Example:** ‘the rector of Masaryk University’
$(((o\iota)\tau)\omega)$, $(o\iota)_{\tau\omega}$	a *property* – an atomic intensional feature that an individual either has or has not according to a chosen world and time. Such feature is then represented by a class (its characteristic function) of those individuals that have the feature in a certain world and time moment. **Example:** ‘being rounded’
$(\xi\tau)$, ξ_τ	a *ξ-chronology* – a mapping that specifies the flow of changes of a ξ-object in time. **Example:** ‘yesterday’ is a time interval of 24 hours that ends at the last midnight, thus it represents a different interval every day.
$((\xi\tau)\omega)$, $\xi_{\tau\omega}$	an *intension* – expresses the dependency of the related ξ-object on the selected possible world and time moment. The application of an intension to the world and time (i.e. composition of a $\xi_{\tau\omega}$-object with argument of type ω and then with argument of type τ resulting in a construction of a ξ-object) is called the *intensional descent.* If ξ is not itself an intension, it is called an *extension* and represents an entity whose value does not change with world and time. All mathematical objects (numbers, operations, axioms) correspond to extensions.

dure of translation of a textual form of a natural language sentence (in Czech) into the corresponding construction of Transparent Intensional Logic, which then describes the logical meaning of the sentence.

In some cases, the resulting sentence construction is not *pragmatically anchored*, i.e. it contains free variables, which reference the pragmatic situation of the discourse as a whole. Before using such sentence in inference, the pragmatic meaning of the sentence must be acquired with filling the values of the free variables that represent personal pronouns, proper names or discourse links. However, the discourse analysis is beyond the scope of NTA and is not further discussed in this text.

In the following sections, we concentrate on the clause construction with the usage of three lexicons:

- the lexicon of tokens
- the lexicon of verb frames
- the lexicon of functional items such as prepositions or conjunctions.

The content and format of these lexicons is exemplified further in the text.

4.2.1 Verb Frame Analysis

The TIL type of the object, that is denoted by a verb in the finite form, can be derived from the actual verb frame instantiated in the sentence. Each of the verb arguments may be assigned a different type from the *lexicon of verb frames*. In the VerbaLex lexicon of verb valencies for Czech (see the Section 2.1), we record the *syntactic surface structure* of the sentence constituents as well as their *semantic role*. During the logical analysis in TIL, we need to identify yet another level of the denotation of a verb argument – its *meaning function*. On this level, we enter the construction of the TIL object represented by the corresponding natural language expression.

The distinction of the three levels of verb frame representation may be demonstrated on the example of the verb ‘brát’[3] with a valency ‘někomu něco.’[4] The three levels then can look like[5]

[3]‘to take’

[4]‘something from somebody’

[5]At this level of analysis, the Agent (subject) is not processed. Its processing is postponed to the clause level analysis.

1. syntactic surface structure:

   brát
   $\text{někomu}_{\text{human.NP, dat., no prep.}}$
   $\text{něco}_{\text{non-human.NP, accus., no prep.}}$

   This level reflects the properties of constituents that can be derived following the morphological and syntactic analysis of the sentence.

2. semantic function:

   `AG<person:1>` brát `PAT<person:1> POS<possession:2>`

   The semantic function denotes the role of the verb arguments in the activity expressed by the verb – PAT, the one that is referenced by the verb as the receiver of the activity, and POS, the PAT's possession that is taken away.

3. meaning function:

   $\mathsf{brát}/(o(o\pi)(o\pi))_{\omega}\iota\iota$
   $x \ldots \iota$
   $y \ldots \iota$: $s_{wt}\mathrm{y}$, $s \ldots (o\iota)_{\tau\omega}$

   On this level, we try to find the construction of the object that is represented by the corresponding constituent – $x \ldots \iota$, a specific individual, and the other $y \ldots \iota : s_{wt}y, s \ldots (o\iota)_{\tau\omega}$, an individual from a class of individuals or an individual with a specified property.

In the analysis of the verb valency frame in the NTA, we need to find the appropriate translation from the syntactic structure to the meaning function. The particular construction and type that appears in the resulting sentence analysis depends on (at least):

1. the actual *input lexical items* the constituent consists of – their analysis has to be found in the *lexicon of tokens.*

2. the *context* – the lexicon often offers more than one possible analysis of the lexical item. However, on the upper level the surrounding lexical items may provide more details to the specification of the subject and so allow to select only the appropriate analyses of the item.

Table 4.2: Examples of a noun analysis.

Noun	Analysis	Description
pes, člověk [*]	$x \ldots \iota$: $\mathbf{pes}_{wt}x$, $\mathsf{pes}/(o\iota)_{\tau\omega}$	an individual from the class of individuals – such x for which $\mathbf{pes}_{wt}x$ holds
prezident [*]	$\mathsf{prezident}/\iota_{\tau\omega}$	an individual role
volitelnost [*]	$\mathsf{volitelnost}/(o\iota_{\tau\omega})_{\tau\omega}$	a property of an individual role
výška, hmotnost [*]	$\mathsf{výška}/(\tau\iota)_{\tau\omega}$	a quantity
výrok, tvrzení [*]	$p \ldots *_\pi$: $\mathbf{výrok}_{wt}p$, $\mathsf{výrok}/(o*_\pi)_{\tau\omega}$	a construction of a proposition from the class of constructions of a proposition
válka, smích, zvonění [*]	$\mathsf{válka}/(o(o\pi))_\omega$	a class of episodes – an activity (with some time persistence) that directly corresponds to an episodic verb
leden, podzim [*]	$\mathsf{leden}/(o(o\tau))$	classes of time moments – time intervals specified by month or season.

[*]'dog', 'human'; 'president'; 'eligibility'; 'height', 'weight'; 'statement', 'assertion'; 'war', 'laughter', 'ringing'; 'January', 'autumn'

The basic guide-post for the list of valencies of Czech verbs that keeps the syntactic structure of the verb valency should route the translation of a valency expression (i.e. a specification of a verb argument) in the following way:

a noun group (with/without preposition).

A noun phrase is usually formed by a core, a noun, which is preceded by adjuncts in the form of an adjective, a pronoun or a numeral or a combination of such items. In the simplest case, the noun phrase consists of just one noun, whose analysis is found in the lexicon. Examples of common analyses of a noun are presented in the Table 4.2.

an adverbial phrase
: The constructions of adverbial phrases usually works as a modifier of the verbal object of the verb and is not described here in detail.

a subordinate clause
: The sentence building includes the description of analysis of relative and other subordinate clauses by means of clause combinations with the conjunction object as its functor.

an infinitive
: The infinitive form of a verb in the position of a verb argument is analysed as the world-instantiated verbal object (object of type $o(o\pi)(o\pi)$) of the corresponding verb.

Following these guidelines, the current lexicon of verb frames provides the system with the information about:

- verb *lemma*
- *surface verb frame* – information about morphological and syntactic features of verb arguments
- the TIL *types of the arguments* of the verbal object
- the *schema* of the verbal object construction

An example of the current version of a lexicon entry for the verb 'brát' with valency frame 'někomu něco' (to take something from somebody) is presented in the Figure 4.1.

For an automatic verb object type assignment based on the VerbaLex lexicon see the detailed description in the Section 4.3, where we show how CVFs describe the surface valencies of verbs (i.e. their respective morphological cases in Czech) as well as the semantics of their predicate-argument structure. Concerning the latter we make use of the deep semantic roles expressed by two-level labels based partly on the Top Ontology (EuroWordNet [VB^{+}98]) and partly on the selected literals from Princeton WordNet [Mil90].

```
; lemma
brát
; encoded surface valency frame with TIL types of arguments (i=ι)
hPc3t{i}-hTc4t{i}
; the verbal object schema
:exists:  V(v):V(v) :and:  V(v)=[[#0,try(#1),try(#2)],V(w)]
```

Figure 4.1: Lexicon entry for the verb frame 'brát někomu něco' (to take something from somebody).

4.2.2 The Sentence Analysis

In this stage, we have information about the logical analysis of a verb group with its arguments and adverbial modifiers. What remains to be specified, is the basic guide-post that suggests the best order in which all the partial analyses of phrases, clauses and sentence should proceed.

We suppose that the input state of the logical analysis is formed by an already disambiguated (uniquely identified) syntactic derivation tree. Hence, the logical analysis may run either after the end of the syntactic analysis of the input sentence, or as well in parallel with it, in which case the necessary procedures perform as certain contextual actions, which work over the possible combinations of the (locally) analysed constituents. The asset of such parallel approach lies in its capability to prune analyses which are type-inconsistent, e.g. if the verb expects an individual as its argument, the type checking mechanism would not allow a proposition to take this place. However, the cases where such pruning may reduce the extent of the syntactic analysis are quite rare (remember, this pruning applies only on sentences which are correct in their syntax but inconsistent in the types of their constituents) or they can be substituted with the verb frame analysis only. The drawback of the parallel analysis lies also in the time and space spent on the overabundant logical analysis of those subtrees that are not part of the resulting derivation tree (i.e., in the parallel analysis, we cannot cast away any subtree that is successful "so far," even if it may be ruled out within the successive analysis).

That is why, we have chosen the logical analysis to run within the fourth group of our meta-grammar actions – the actions based on derivation tree.[6] The process proceeds in certain (time) successive steps, which are summarized in the following paragraphs.

4.2.2.1 Lexicon of Tokens

The logical analysis starts to build the construction of the whole sentence from *inside*, i.e., in concordance with the Compositionality Principle, the meaning of the compound is constructed as the meaning of its constituents. Therefore, the first step must necessarily run in the lowest part of the derivation tree – the analysis of the input lexical items. In this step, we have not much choice other than to look up the proper analysis (analyses) of the lexical items in the *lexicon of tokens*. An example of entries in the lexicon of tokens is:

> /* *type definition:* $(o\iota)_{\tau\omega}$ *= property (of an individual)* */
> `#T_PROP (((oi)t)w)`
>
> /* *lemma* */
> `pečený`
> /* *schema – object trivialization,* ^{0}pečený ... $(o\iota)_{\tau\omega}$ */
> `O(pečený/T_PROP)`
>
> /* *lemma* */
> `kuře`
> /* *schema – property ascription,* $(\exists i \ldots \iota)[^0\mathsf{kuře}_{wt}\ i]$ */
> `:exists:  V(i/i):V(i) :and:  [awt(O(kuře/T_PROP)),V(i)]`

Here, the word 'pečený' (roasted) is analysed as a property of individuals and the word 'kuře' (chicken) is given the analysis of an individual bearing a specific property (to be a chicken).

The lexicon can supply some wild-card values based on the grammatical category of the lexical item, but in such case we risk the possibility of incorrect type assignment (e.g. the word 'výška' (height) cannot be analysed as an individual). Hence, as a result of this part, we receive the type of each lexical item as well as a schema of its working with other (dependent) constituents (e.g. a conjunction is accompanied with the schema of the relevant clause (propositions) as its arguments). Such a lexical item that expects some arguments to be meaningful is called a *functional* lexical item.

[6]see the Section 3.2.3.

```
prep_noun_phrase -> prep noun_phrase
    agree_case_and_propagate($1, $2)
    depends($1,$2)
    add_prep_ngroup($2)
    rule_schema($@,"lwt([awt(#1),try(#2)])")

noun_phrase -> left_modif noun_phrase
    agree_case_number_gender_and_propagate($1, $2)
    depends($2,$1)
    rule_schema($@,"lwtx(awtx(#1) and awtx(#2))")
    rule_schema($@,"lwtx([[awt(#1),#2],x])")
```

Figure 4.2: Examples of meta-grammar rules for prepositional noun phrase and noun phrase with multiple rule schemata. The actual choice of the proper rule schema is guided by the type checking system during the running time of the analysis.

4.2.2.2 Rule Schemata

The analysis then moves up the derivation tree, rule by rule. Each rule is supplemented with a similar schema as the functional lexical items, a schema that tells how the constituents, that correspond to the nonterminals (or preterminals) on the right hand side, combine together to form a construction of the left side nonterminal. For an example of the rules with rule schemata see the Figure 4.2. The result of the application of the schema is then subject to the type checking mechanism which safeguards that the constituents agree in the logical type with the others in the resulting construction, i.e. that all arguments of a composition have the types needed by the corresponding function.

In this way, we form the constructions of constituents such are noun phrases or adverbial phrases up to the level of a clause.

4.2.2.3 Clause Construction

In a rule of the form '`clause` $\rightarrow$ `...`', the process becomes a little more complex than to be described in one step only. In such rule, we have identified the kind of the verb group, i.e. whether it is an attributive or an episodic verb,[7] active or passive voice and past, present or future tense. In groups of so called intersegments we have the candidates for the verb arguments and free adjuncts in the form of noun phrases, prepositional noun phrases, other clauses or adverbial phrases. In several successive steps, we now need to form the construction corresponding to this particular clause:

1. first, we try to identify the *subject* (typically Agent) of the clause. In Czech, we can seek for a noun phrase (including a single adjectival group, optionally followed by an indeclinable word such as particle, adverb or interjection) in nominative.

   If the subject cannot be determined, we suppose that it is unexpressed[8] and supply a indefinite subject of the type of individual or a class of individuals according to the number (singular or plural) of the verb.

2. after that, we look up the *finite form verb* in the lexicon where we obtain all acceptable verb frames of this verb with the corresponding analyses (that includes the types of the verb arguments, as well).

3. now, we need to determine the verb arguments in the sentence – the process is similar to the procedure in the Section 4.2.2.2 above. In order to reduce the multiplicative extent of the number of participants to be checked during this process, we run one round of pruning yet before we start to build the construction – we check all the intersegments against the available verb frames and first, score out those that with certainty cannot take part in the verb frame, and secondly, check all the possibilities (based only on the stated grammatical categories) of their fitting in place in the verb frame (e.g. we do not allow two independent verb objects in accusative). After this, we obtain the possible verb arguments that are then type checked according to the requirements of the verb.

4. if we have linked in a relative clause or a clause with an inexplicit subject, we try to supplement it with the subject of the principal

[7]see the Section 4.3 for description of attributive and episodic verbs.

[8]In Czech, this is a frequent case – if the sentence subject is a personal pronoun (I, you, he, she, ...) then the subject can be completely left out (*unexpressed*) since people are able to recognize it from the ending of a finite verb in the sentence.

clause (i.e. if its verb and the subject agree in number and gender). Otherwise, we find the clause's subject as unexpressed.

In this way, we obtain the construction of a clause.

4.2.2.4 Whole Sentence Construction

Eventually, we process the clauses' constructions according to their conjunction. The details of their combinations in the sentence building process can be found in [Hor02].

Thus, following these steps and the guidelines provided in the previous section, we can accomplish the logical analysis of the whole natural language sentence.

4.3 Sentence Logical Analysis Using Complex Valency Frames

In this section we describe the translation of VerbaLex CVFs into constructions of verb phrases, which form a core of sentence logical analysis. Valency frames have been built in several projects (VALLEX for Czech PDT [Žab05] or VerbNet [KKRP06]). Motivation for the VerbaLex project came from comparing Czech WordNet verb frames with VALLEX. The main goal of VerbaLex is an automatic processing of verb phrases exploiting explicit links to Princeton WordNet. The complex valency frames we are working with can be characterized as data structures describing predicate-argument structure of a verb which contains the verb itself and the arguments determined by the verb meaning. The argument structure also displays the semantic preferences for the arguments. On the syntactic (surface) level the arguments are most frequently expressed as noun or pronominal groups in one of the seven cases (in Czech) and also as prepositional cases or adverbials.

The semantics of the arguments is typically labeled as belonging to a given semantic role, which represents a general role plus subcategorization features (or selectional restrictions). Thus valency frames in VerbaLex include information about:

1. the syntactic (surface) information about the syntactic valencies of a verb, i.e. what morphological cases (direct and prepositional ones

in highly inflected languages such as Czech) are associated with (required by) a particular verb,

2. semantic roles that represent the integration of the general labels with subcategorization features required by the meaning of the verb.

The inventory of the semantic roles is partly inspired by the Top Ontology and Base Concepts as they have been defined within the EuroWordNet project [VB^{+}98]. Thus we work with the general or "large" roles like AG(ENT), ART(IFACT), SUBS(TANCE), PART, CAUSE, OBJ(ECT) (natural object), INFO(RMATION), FOOD, GARMENT, VEHICLE and others. These first level roles are combined with the literals from Princeton WordNet 2.0 [WN07] where literals represent subcategorization features allowing us to climb down the hypero/hyponymic trees to the individual lexical units.[9]

The verb entries are linked to the Czech and Princeton WordNet 2.0, i.e. they are organized around the respective lemma in synsets (synonymical sets) with numbered senses. The inventory of the semantic roles we work with clearly represents a sort of ontology which tries to cover word stock of Czech verbs and can be used as a base for a semantic classification and subclassification of the verbs. The ontologies represent theoretical constructs designed from the "top" and as such they are not directly based on the empirical evidence, i.e., corpus data. Thus there is a need to confront the ontologies and the inventories of the semantic roles that can be derived from them with the corpus data and see how well they can correspond to them. For this purpose we are experimenting with the corpus data obtained from the Word Sketch Engine [KRST04].

The Transparent Intensional Logic comes with a dissociation of significant verbs into two groups according to the classification of their meaning:

1. by *attributive verbs* we ascribe qualities or properties to objects. Attributive verbs are typically expressed by the respective form of the verb "to be" combined with an expression denoting a property; examples: "to be red" or "to be mellow" or with a general substantive like "to be a traitor," "to be a tree."

[9]see the Section 2.1.2.

2. *episodic verbs*, on the other hand, specify actions performed by a subject. An episodic verb does not describe its subject's state in any moment of time, it rather describes an episode of doing something at the certain time moment (and necessarily some time before that moment plus the expectation that it will last also in the next few moments, at least). TIL provides a complex handling of episodic verbs including the verb tense, aspect (perfective/imperfective) or active/passive state. All these features are concentrated around the so called verbal object, the construction of which (i.e., the meaning of a particular verb phrase) is the application of (the construction of) the verb to (the constructions of) the verb arguments.

Since the analysis of attributive verbs is usually quite simple, we will concentrate in the following text on examples of selected episodic verbs from VerbaLex and their logical analysis using the complex valency frames.

The TIL type of episodic verbal objects is $(o(o\pi)(o\pi))_\omega$, where π is the type of propositions $(o_{\tau\omega})$. See [Tic80, Hor02, pp. 64-73] for detailed explanation. Our analysis is driven by a linguistic (syntactic) context that signals the semantic fact that there is always a function involved here, so that we have to ascribe types to verb arguments.

4.3.1 Examples of Logical Analysis

In this section, we have chosen 7 verbs with their verb frames from VerbaLex and we will use them as examples of the algorithm for determining the verb type in the TIL logical analysis procedure.

dát (give)
dát:2 / dávat:2 / darovat:1 / věnovat:1 (give:8, gift:2, present:7)
give as a present; make a gift of
–frame: `DON<organization:1>`$^{\text{obl}}_{\text{what\_nom}}$ `VERB`$^{\text{obl}}$
`OBJ<object:1>`$^{\text{obl}}_{\text{what\_acc}}$ `BEN<person:1>`$^{\text{obl}}_{\text{to\_whom\_dat}}$
–example: firma věnovala zaměstnancům nová auta (a company gave new cars to the employees)
–use: prim

The verb arguments in this frame are: *who*, *to whom*, *what* (all obligatory) with (at least) two options: a) *to whom* is an individual, b) *to* *whom* is a class of individuals. The respective verb types are ad a): $((o(o\pi)(o\pi))_\omega \iota\iota\iota)$, ad b): $((o(o\pi)(o\pi))_\omega \iota(o\iota)\iota)$.

For example, take *to whom* = "to the employees of a given institution." Then "to be an employee of the institution XY" is a property, say $\mathsf{Z}/(o\iota)_{\tau\omega}$. So "The company gave to the employees of XY...," not taking into account grammatical tenses and omitting trivializations we get $\lambda w\lambda t[\mathbf{Give}_{wt} XY\, \mathbf{Z}_{wt}\ \ldots]$.[10]

With this example, we can show that CVFs are used not only for determining the verbal object type, but also for stating additional prerequisities (necessary conditions) for the sentence constituents. The full analysis using the verb frame above thus contains, except the verb phrase part, the conditions saying that "X gives Y to Z $\wedge$ organization(X) $\wedge$ object(Y) $\wedge$ person(Z)." The predicates *organization*, *object* and *person* here represent the properties denoted by the corresponding terms in the WordNet hypero-hyponymic hierarchy.

dát:15 / dávat:15 / nabídnout:3 / nabízet:3 (give:37)
offer in good faith
–frame: $\texttt{AG<person:1>}^{\text{obl}}_{\text{who\_nom}}$ $\texttt{VERB}^{\text{obl}}$
$\texttt{ABS<abstraction:1>}^{\text{obl}}_{\text{what\_acc}}$
$\texttt{REC<person:1>}^{\text{obl}}_{\text{to\_whom\_dat}}$
–example: dal jí své slovo (he gave her his word)
–example: nabídl jí své srdce (he offered her his heart)
–use: fig

Here we have an idiom ("to give word", i.e. "to promise"), which corresponds to an (episodic) relation between two individuals. Therefore the type of the verb is $((o(o\pi)(o\pi))_\omega \iota\iota)$, the second ι corresponds to *to whom.*

bránit (prevent)
bránit:1 / zabránit:2 / zabraňovat:2 / zamezit:2 / zamezovat:2 (prevent:2, keep:4)
prevent from doing something or being in a certain state
–frame: $\texttt{AG<person:1>}^{\text{obl}}_{\text{who\_nom}}$ $\texttt{VERB}^{\text{obl}}$
$\texttt{PAT<person:1>}^{\text{obl}}_{\text{to\_whom\_dat}}$
$\texttt{ACT<act:2>}^{\text{obl}}_{\text{inf}}$

[10] XY has the type ι here, being a collective rather than a class.

–example: zabránila mu uhodit syna (she prevented him from hitting the son)
–use: prim

bránit:1 / zabránit:2 / zabraňovat:2 / zamezit:2 / zamezovat:2 (prevent:2, keep:4)
prevent from doing something or being in a certain state
–frame: $\texttt{AG<institution:1>}^{\text{obl}}_{\text{what\_nom}}$ $\texttt{VERB}^{\text{obl}}$
$\texttt{PAT<person:1>}^{\text{obl}}_{\text{to\_whom\_dat}}$ $\texttt{ACT<act:2>}^{\text{opt}}_{\text{in\_what\_loc}}$
–example: policie mu zabránila v cestě do zahraničí (police prevented him from travelling abroad)
–use: prim

Here, arguments of the verb correspond to the phrases *who*, *to whom*, *in* (*from*). The third argument has the type of an activity given, of course, by an episodic verb "hit the son," "travel abroad" (the substantive form travelling abroad can be construed as that activity). The type of the verb is $((o(o\pi)(o\pi))_\omega \iota((o(o\pi)(o\pi))_\omega))$.

říct (say)
říct:1 / říkat:1 / říci:1 / pravit:1 (say:6)
utter aloud
–frame: $\texttt{AG<person:1>}^{\text{obl}}_{\text{who\_nom}}$ $\texttt{VERB}^{\text{obl}}$
$\texttt{COM<speech act:1>}^{\text{obl}}_{\text{what\_acc,that,dsp}}$
$\texttt{ADR<person:1>}^{\text{opt}}_{\text{to\_whom\_dat}}$
–example: říct kolegovi dobrý den (say hello to a colleague)
–example: řekl, že to platí (he said that it holds)
–example: pravil: "Dobrý den" (he said: "Good day")
–use: prim

The case questions for the corresponding arguments of the verb "říct" are a) *who*, *what*$_1$, b) *who*, *what*$_2$, c) *who*, *to whom*, *what*$_1$, and d) *who*, *to whom*, *what*$_2$. Examples of instantiated sentences can be a) "Charles says 'Hello'," b) "Charles says that he is ill," c) "Charles says to his colleague 'Hello'," or d) "Charles says to his colleague that he is ill."

The quotation context (ad a), c)) is normally impossible to type. Unless we want to go into some deep analyses we can ascribe to any quoted expression the type of individual. The relation to and unquoted subordinate clause is analysed as a general construction of type $*_n$. The resulting types of verbs are then

a) $((o(o\pi)(o\pi))_\omega \iota\iota)$,
b) $((o(o\pi)(o\pi))_\omega \iota *_n)$,
c) $((o(o\pi)(o\pi))_\omega \iota\iota\iota)$,
d) $((o(o\pi)(o\pi))_\omega \iota\iota *_n)$.

brečet$_1$ (cry$_1$) because of something, for something
brečet:1 / plakat:1 (cry:2, weep:1)
shed tears because of sadness, rage, or pain
–frame: `AG<person:1>`$^{\text{obl}}_{\text{who\_nom}}$ `VERB`$^{\text{obl}}$
`CAUSE<cause:4>`$^{\text{obl}}_{\text{due+to+what\_dat,over+what\_ins, for+what\_acc}}$
–example: brečela kvůli zničeným šatům (she cried for spoiled clothes)
–example: plakal nad svou chudobou (he cried over his poverty)
–example: plakal pro své hříchy (he cried for his sins)
–use: prim

brečet$_2$ (cry$_2$) for somebody
brečet:1 / plakat:1 (cry:2, weep:1)
shed tears because of sadness, rage, or pain
–frame: `AG<person:1>`$^{\text{obl}}_{\text{who\_nom}}$ `VERB`$^{\text{obl}}$
`ENT<person:1>`$^{\text{obl}}_{\text{for+whom\_acc}}$
–example: plakala pro milého (she cried for her boy)
–use: prim

If I “cry because of, for” etc., then the role of causing is played by this *because of*. Crying is an episodic verb, whereas *because of* is a relation between propositions, often between events. We have therefore because of/$(o\pi\pi)_{\tau\omega}$, where the first π ($= o_{\tau\omega}$) belongs to the proposition denoted, e.g., by “clothes have been spoiled” or that “the respective individual is poor, sinful” etc., and the second π to the proposition that the respective individual cries.

In case of “to cry for somebody” the respective type is again a relation $((o(o\pi)(o\pi))_\omega \iota\iota)$, although this *for* hides some cause, which is, however, not mentioned.

With this verb, we will describe the analysis of verb entailment handling in TIL. If we analyse a general case of the above mentioned meanings of *cry* (cry$_1$ – because of something, cry$_2$ – for somebody) simply "to cry" (cry$_3$, "He cries all the time"), this verb type is a verbal object without arguments, $(o(o\pi)(o\pi))_\omega$. In addition to this the following rule holds: If X cries because of ... or X cries for ..., then X cries. In this way the semantic dependence between the three cases of crying is given; otherwise we would not be able to detect this connection, e.g. between cry$_1$ and cry$_2$.

absolvovat (undergo)
absolvovat:2 / prožít:1 / prožívat:1 (experience:1, undergo:2, see:21, go through:1)
go or live through
–frame: `AG<person:1>`$^{\text{obl}}_{\text{who\_nom}}$ `VERB`$^{\text{obl}}$
`EVEN<experience:3>`$^{\text{obl}}_{\text{what\_acc}}$
`LOC<location:1>`$^{\text{opt}}_{\text{in\_what\_loc}}$
–example: absolvoval vyšetření na psychiatrické klinice (he went through investigation in a psychiatric clinic)
–use: prim

In general "absolvovat něco (undergo something)" is an episodic relation to an *event* (type π),[11] so the verb type is $((o(o\pi)(o\pi))_\omega \iota\pi)$. In some cases we may also use a relation to an episode (specific class of events, type $(o\pi)$), then the type is $((o(o\pi)(o\pi))_\omega \iota(o\pi))$, and "investigation in a clinic" has to be defined as a sequence of events.

akceptovat (accept)
akceptovat:3 / přijmout:6 / přijímat:6 (accept:4)
react favorably to; consider right and proper
–frame: `AG<person:1|social group:1>`$^{\text{obl}}_{\text{who\_nom}}$ `VERB`$^{\text{obl}}$
`STATE<state:4>|EVEN<event:1>|INFO<info:1>`$^{\text{obl}}_{\text{what\_acc}}$
–example: akceptujeme jeho povahu (we accept his character)
–example: lidé přijali nový zákon s nadšením (people accepted new law with enthusiasm)
–use: prim

[11] see [Tic80, Hor02, p. 65].

We can "accept" nearly anything. Here we meet the problem of type-theoretical polymorphism, which is handled here as a polymorphic type scheme $((o(o\pi)(o\pi))_\omega \iota\alpha)$, for an arbitrary type α. A quintessence of such a polymorphism: "think on (about)" – one can think of an object of any kind.

učit (teach)
naučit:1 / učit:1 / poučit:1 / poučovat:1 / vyučovat:1 (teach:1, learn:5, instruct:1)
impart skills or knowledge to
–frame: `AG<person:1>`$^{\text{obl}}_{\text{who\_nom}}$ `VERB`$^{\text{obl}}$
`PAT<person:1>`$^{\text{opt}}_{\text{whom\_acc}}$
`KNOW<subject:3>`$^{\text{obl}}_{\text{what\_acc,to\_what\_dat}}$
–example: naučil dítě abecedu (he educated a children in the alphabet)
–example: učí studenty matematiku (he/she teaches mathematics for students)
–example: vyučuje dějepisu (he/she teaches history)
–use: prim

If understood as in "What does (s)he live off? (S)he teaches." it is the case of cry$_3$ (see above). "To teach" understood as in "He teaches history, maths", etc., the analysis depends on which type is given to the school subjects, disciplines. One possibility is to analyse them as properties of a set of propositions, $(o(o\pi))_{\tau\omega}$. Then to teach receives the type $((o(o\pi)(o\pi))_\omega \iota(o(o\pi))_{\tau\omega})$. If "teaches alphabet" is the case then we have to decide what we mean by "alphabet." Here the point is "to teach (learn) to associate symbols and sounds (phonemes?)," so the respective type of alphabet is $(\alpha\beta)$, where α is the type of symbols, β the type of sounds. In the analysis of "to educate somebody in something" the verb takes an individual as its additional argument: $((o(o\pi)(o\pi))_\omega \iota\iota\alpha)$, where α is the type of the discipline.

In all the examples, we have displayed the relations between the two-level semantic roles used in the VerbaLex verb frames and the resulting logical analysis types of the verbal object as the main part of the clause's logical construction. The algorithmization of this procedure uses a list of all roles used in the lexicon (there are about 200 roles used) with the corresponding (ambiguous) logical types of the constituents. In this way we can form a basic skeleton of the automatic translation of text to logical constructions.

4.4 TIL Knowledge Base Representation

We can look at the knowledge base (KB) design as consisting of *the ontology* and the description of the way how the *logical inference* is performed. The ontology is used as a specification of all concepts, that the inferring machine knows about. Only those concepts that are committed to the ontology can ever be thought of as being processed by the machine when used in a question-answering or a dialogue system. In our case the ontology is well defined by means of TIL. All objects that are stored in the knowledge base are derived directly from input constructions that reflect the meaning of facts that the user had implanted to KB. Every object has an equivalent type in the epistemic framework of TIL.

The system knowledge base design is a form of a semantic network consisting of declarative memory for storing facts and procedural memory for storing inference rules (a similar approach is used, e.g., in the Soar system [RLN93]). Facts in the declarative memory are strictly time--dependent, which means that each fact is bounded to the time moment, at which the fact was stored.

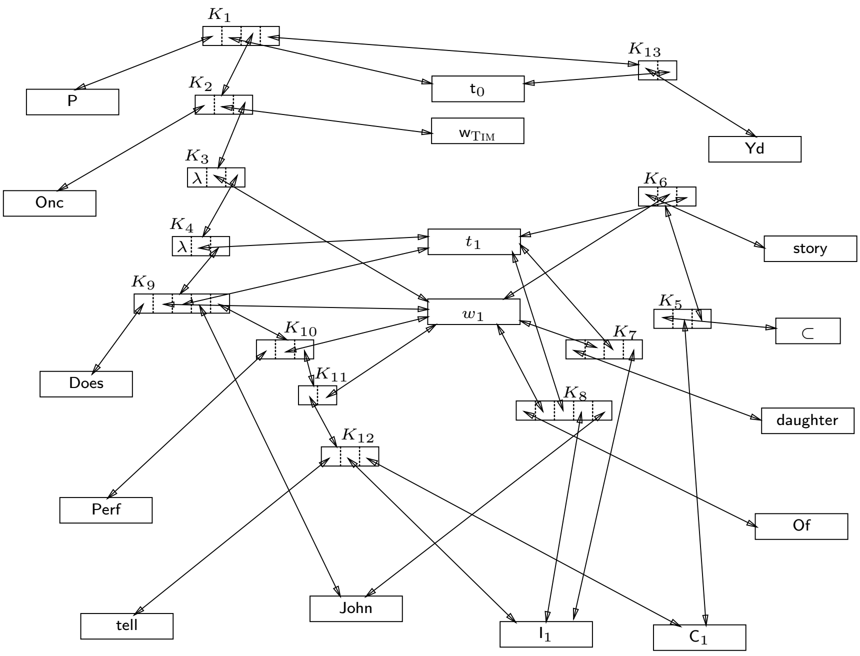
The KB structure can be illustrated by the example representing the facts that arise from the input sentence

John told a story to his daughter yesterday. (4.2)

The equivalent TIL construction is stated in the Figure 4.3.

When storing the facts in KB, the existentially quantified variables are replaced with newly allocated constants of the appropriate type (by the process of skolemization) or linked to already allocated objects.

The part of KB that represents our construction consists of 15 simple type objects (objects without subconstructions), 2 variables (w_1 and t_1) and 13 structured terms ($K_1, \ldots, K_{13}$). The whole semantic network that represents this sentence is depicted in the Figure 4.3. The objects I_1 and C_1 are the constants that replaced the skolemized variables x and z. Two other constant objects t_0 and $\mathsf{w}_{\mathrm{TIM}}$ refer to the moment of the utterance of (4.2) and the reference world of the TIL inference machine. Those



$$\lambda w \lambda t \left[\mathbf{P}_t \left[\mathbf{Onc}_w \lambda w_1 \lambda t_1 (\exists x)(\exists z) \left[\mathbf{daughter}_{w_1 t_1} x \wedge \left[\mathbf{Of}_{w_1 t_1} x \, \mathbf{John} \right] \wedge \right.\right.\right.$$
$$\left.\left.\left. \wedge \left[\mathbf{story}_{w_1 t_1} z \right] \wedge \left[\mathbf{Does}_{w_1 t_1} \mathbf{John} \left[\mathbf{Perf}_{w_1} [\mathbf{tell} \, x \, z]_{w_1} \right] \right] \right] \right] [\mathbf{Yd} \, t] \right] \tag{4.1}$$

Figure 4.3: The part of KB with the construction of the sentence (4.2).

constants were put into the construction (4.1) in place of the abstracted variables w and t.

Such format of the knowledge base properly reflects the *reuse* property of constructions and subconstructions. This property is necessary for large knowledge databases, since they share the space of identical constructions, and on the other hand, it helps to identify all the constructions working over one topic (expressed by a subconstruction) in a very straightforward way.

4.4.1 Knowledge Base Implementation – the Dolphin System

Dolphin is a part of a complex project employing TIL to develop a computer system with data understanding ability. Such system will be able not only to store natural language structures but also to mine new facts logically resulting from input and the data stored before. One of the final benefits of the whole system will be no need for special human-machine communication language – the user will just ask and Dolphin will answer what can be inferred from the knowledge base to the question. The Dolphin system is thus an experimental implementation of the TIL knowledge base developed by Andrej Gardoň in the MU NLP Centre under the supervision of Aleš Horák.

The Dolphin system is designed to process the output of the syntactic parser `synt` (described in the Chapter 3), which is able to produce syntactic trees as well as logical representations of input sentences in the form of TIL constructions. In Dolphin, the constructions are parsed and stored in the knowledge base (KB), which takes the form of a semantic network. Currently, the Dolphin system stores labels of natural language objects described in the Czech language. However, the actual chosen language does not limit the functionality of the system. The Czech language is used in Dolphin because the transcription of natural language sentences to TIL constructions is handled by the Normal Translation Algorithm (NTA) system [Hor02] that presently provides only the Czech transcription ability.

When a new input is to be stored in the database, it has to pass through several parts of the system. The Figure 4.4 demonstrates the process of storing new data to Dolphin.

In the first phase of storing a new fact saying "*Lemon is yellow,*" the

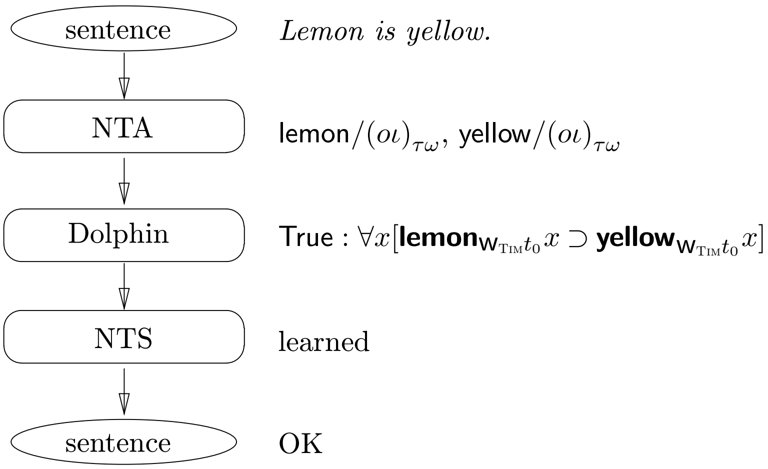

sentence
Lemon is yellow.
NTA
$\mathsf{lemon}/(o\iota)_{\tau\omega}$, $\mathsf{yellow}/(o\iota)_{\tau\omega}$
Dolphin
$\mathsf{True} : \forall x[\mathbf{lemon}_{\mathsf{w}_{\text{TIM}} t_0} x \supset \mathbf{yellow}_{\mathsf{w}_{\text{TIM}} t_0} x]$
NTS
learned
sentence
OK

Figure 4.4: The phases of data flow in Dolphin.

sentence is translated by NTA to the TIL construction of the sentence. Dolphin then parses the construction and stores it in the knowledge base. At the level of propositions, Dolphin interprets logical operators and divides the sentence to several basic facts. The composed construction is stored as a connection among sub-constructions. The last symbol of the sentence (`.`, `?`, `!`) indicates the mode of the database operation (storing/answering/ordering). NTS stands for the Normal Translation Synthesizer and its role is to produce a sentence from TIL descriptions provided by the output of Dolphin.

4.4.1.1 The Dolphin Storage

As we mentioned before, the chosen input natural language does not influence the functionality of the Dolphin KB. On the other hand it is very important in searching for objects. The Dolphin KB thus stores the data in two layers: the *language layer* and the *object layer*.

The basic Dolphin idea is a separation of the language layer from the logical (object) layer. Human words are just a way how to describe some objects. There are many languages but all of them work over the same set of objects. The Dolphin input processing lies in transforming its teacher's words to the KB objects mentioned above. Here `synt` plays its role as it produces TIL transcription of a sentence. E.g., Dolphin after obtaining *apple* on its input takes its TIL description and creates a new object in its object layer – let it be the object #1. Next time we mention this particular *apple*, Dolphin looks up the object #1 in its KB and whatever is told about it is linked to this object.

The Language Layer This layer provides the language encoding ability of the Dolphin system allowing the rest of the system to be completely language independent.

The implementation of the language layer (LL) has the form of multilingual dictionary providing ILIs (inter lingual indexes) for the object layer. Each LL record contains an object name expressed by words in different languages.

The Object Layer This layer is quite complex so only basic features are mentioned, see [Gar07] for details. Each TIL object is given a type from the type hierarchy built over four basic types – ι (individuals as object labels), o (truth values, True and False), τ (time moments and real numbers) and ω (possible worlds).

One of the essential KB objects for inferring the object properties are the objects of types including ι in it. Thus those objects are stored in separate files containing this information:

- the type of the object
- the connection to the language layer, provides naming of objects using words of natural language
- membership – many objects form classes or relations. To provide fast way of searching for data, it is essential to know in which classes/relations an object has a membership
- members – only for objects with composed type to identify simple objects that form them

Objects of type τ and ω are currently just stored in the database without any special handling.

4.4.1.2 The Input Sentence Processing

Let us take an example input sentence (in Czech, as this is the current input language of `synt`):

Jablko je červené. (An apple is red)

and its corresponding TIL transcription:

$$\lambda w_1 \lambda t_2 (\exists i_3)([{}^0\mathsf{jablko\text{-}0}_{w_1 t_2}, i_3] \;\wedge\; [{}^0\mathsf{červený\text{-}2}_{w_1 t_2}, i_3]) \ldots \pi$$

At first, variables are identified and single applications are isolated. Variables that are not covered by λ-abstraction are replaced by new constants named after the variables. The λ-abstracted variables are currently handled in a simplified way – when the λ-abstraction is used without quantification, the abstracted variables usually go over the ω (possible worlds) or τ (time moments) types. In case of an ω-variable, the Dolphin's world $w_{Dolphin}$ is assigned to it. All time variables (of type τ) are, in the current version, replaced by an object representing a "general time independent object," so as we can see the current version does not support time processing at all. This feature will be the main goal of our future work. Other quantified variables are initialized according to the system running mode. There are two running modes currently available – the *learning mode* and the *question answering mode.*

The Learning Mode processes and stores new facts and it is activated by a full stop mark at the end of a sentence. In this mode, each uninitialized variable is replaced by a new object named after the variable. If we want to add new facts about an object previously mentioned in the conversation with Dolphin, we have to stress this with the demonstrative pronoun (*ten*, *ta*, *to* – in Czech this corresponds to the definite article in English). The object is then marked with an exclamation mark (!) and is looked up in the knowledge base and the variable is replaced by the stored object. For example if we want to add the fact that

To jablko je červené. (The apple is red.)

the *apple* is analysed as ${}^0\mathsf{!jablko\text{-}0}/(oi)_{\tau\omega}$ and then found as the previously stored object #23 and the construction would thus contain object ^{0}#23 instead of the i_3 variable displayed above. Currently, the system supports only the existential quantification since the universal quantification defines new inference rules and the complex inference has not been implemented so far.

We may mentioned the way, how possible fact conflicts are handled – if the input fact assigns something to an existing KB object and the fact is in conflict with the KB content then an error message is raised. For demonstration let us have a sentence

The apple is red.

stored in KB. Now we would like to store the sentence

The apple is not red.

This raises an error message and the second sentence is not stored. Today there is no tool for saying to Dolphin:

A fact in KB is wrong, I have the correct one.

so whatever is stored in KB will remain unchanged until a reset of the whole database.

The Question Answering Mode works similarly to the learning mode but there is no unification of free variables with new objects. Instead, the first application containing an uninitialized variable suggests a way how to unify this variable with a particular value. If we take our example sentence as a question

Je nějaké jablko červené? (Is there any red apple?)

first application containing free variable i_3 (variables w_1 and t_2 are initialized as described above) will be

$$[{}^0\mathsf{jablko\text{-}0}_{w_1 t_2}, i_3]$$

Thanks to this application, i_3 is unified with an object that is stored in Dolphin and is an Apple (is linked to the class construction of apples). The following application (saying that the object is *red*) posts a second requirement to check. If the object unified with i_3 is not Red (there is no relation between the unified object and the class of red objects), the system returns to the state when i_3 was initialized and tries another possibility. If all possible ways were checked and there is no object that is Apple and Red at the same time, the answer *NO* is returned. In case all the requirements are fulfilled, the answer *YES* is returned with the selected i_3 value. Again there is no universal quantification feature yet, but simple questions such as

Jaké červené objekty znáš? (What red objects do you know?)

can be processed correctly.

4.4.1.3 How the Objects are Stored

Objects are essential elements of logical analysis and the TIL construction processing. In the Dolphin knowledge base, all data are objects related with each other. It does not matter if we store an individual or a function in KB, we are still working with one object. For example, the class of apples (Apple) is represented as the characteristic function of the class, i.e. as a (world and time dependent) function which returns an o-object (boolean *true/false*) for each ι-object (an individual) given as its argument. Each such ι-object is stored separately from Apple, of course.

There is an information that Apple and the corresponding ι-objects are in relation and this information is shared among all participants. Each object is represented as a separate file. In the future, a specialized file system technology can be developed or archiving methods can be incorporated in the storage process without the database functionality limitation or reimplementation.

4.4.1.4 A Complex Example

Let us take three example sentences:

1. *Toto je jablko. (This is an apple.)*
2. *Tato kostka je červená. (This cube is red.)*
3. *To jablko je červené. (The apple is red.)*

and their TIL transcriptions provided as inputs to Dolphin, step by step:

$$\lambda w_1 \lambda t_2([{}^0\mathsf{jablko\text{-}0}_{w_1 t_2}, Toto]) \ldots \pi$$

The word *toto* (this) makes the resulting construction an open construction (with the *free* variable *Toto*) which needs to receive so called *pragmatic anchor*.[12] This input creates a new object in the knowledge base and Dolphin prints the assigned object number:

```
> Toto je jablko. (This is an apple.)
> stored as object 6.
```

The second sentence

```
> Tato kostka je červená. (This cube is red.)
```

with the corresponding TIL construction

$$\lambda w_1 \lambda t_2([{}^0\textsf{!kostka-0}_{w_1 t_2}, i_3] \;\wedge\; [{}^0\textsf{červený-2}_{w_1 t_2}, i_3]) \ldots \pi$$

where the variable w_1 is unified with object #2 (representing the Dolphin's world), the variable t_2 is unified with object #3 (representing the General time object) and the variable i_3 is now initialized with a new object of type ι (let it be #7), since no previously mentioned object from the class kostka was found.

Each trivialization asks the language layer whether it knows the word. If the word is not found, it is stored and a new object is created with a connection to this word.

So $\wedge$ (AND) in our transcription causes that a look-up in the language layer is performed and the object #10 is returned (we suppose that AND was previously stored).

Applications are represented as relations among objects that are participating in the application. Thus in Dolphin, the partial application

$$[{}^0\textsf{červený-2}/(oi)_{\tau\omega} w_1]$$

is represented as a relation between červený and the object of the variable w_1. The information about the relation is stored both in the červený object and w_1 object and of course in the final object – the result of this application is a new $(o\iota)\tau$-object (a *chronology* of the class of objects which are červený/red) which is than applied on the general time object. The final object of the application

$$[{}^0\textsf{červený-2}_{w_1 t_2}, i_3] \ldots o$$

[12]see [Hor02, the Section 5.2.4] or [Mat98, the Section 7.1]

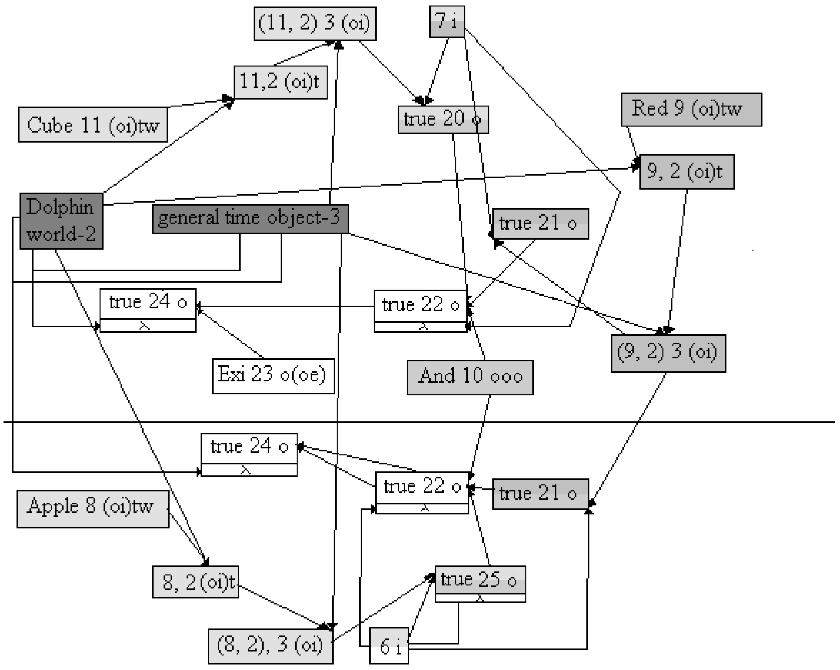


Figure 4.5: The semantic network for the 3 example sentences.

is then created as an o-object (a truth value) and in the learning mode the object receives the value of True.

The third example sentence is stored similarly as the second one with the i_3 variable definition difference.

```
> To jablko je červené. (The apple is red.)
```

TIL transcription

$$\lambda w_1 \lambda t_2([{}^0\text{!jablko-0}_{w_1 t_2}, {}^0\#6] \;\wedge\; [{}^0\text{červený-2}_{w_1 t_2}, {}^0\#6]) \ldots \pi$$

The semantic network for three sentences is displayed in the Figure 4.5. Note that there are many objects with the value of True in the network. This is because each application leads to a unique object and if this object is not yet in KB, it is created. Thanks to this, the λ-abstraction can be done in a straightforward way. It is worth saying that negation is a special function over the *o*-object that does not create a new object but it replaces the existing True value with False. If we ask the question

Je to jabko červené? (Is the apple red?)

TIL transcription has the form of a match:

$$x \ldots o: \quad ([{}^0\mathsf{!jablko\text{-}0}_{w_{Dolphin}t_{now}}, {}^0\mathsf{\#6}] \wedge \\ \wedge [{}^0\mathsf{červený\text{-}2}_{w_{Dolphin}t_{now}}, {}^0\mathsf{\#6}]) \ldots o$$

Basically it asks whether the previously mentioned object #6 is in relation with Apple and Red at the same time. To answer this, the system obtains the result of the applications on the right side of the match and checks whether the final object of the right side is True. In this case, the answer is *YES*. But what happens if we have the question

Je to jablko zelené? (Is the apple green?)

Since we are in the question answering mode, the basic trivialization of zelený (green) will fail as we do not have such object in KB yet. The system does not follow the predicate logic *closed world assumption*, thus the answer is "I DO NOT KNOW" instead of "NO." The question

Které červené objekty znáš? (What red objects do you know?)

is again analysed as a match

$$s \ldots (oi) : {}^0\mathsf{červený\text{-}1}_{w_{Dolphin}t_{now}} \ldots (oi)$$

Here the system evaluates all possible objects which are stored in KB as related to the class červený (applied on the Dolphin's world and the general time object) obtaining the class of all red objects known to Dolphin. Thanks to the system design, this operation consists in fast searching through the semantic network. As can be seen in the Figure 4.5, the Red object applied on the Dolphin's world and the General time object ((*oi*)-object with `(9,2) 3` label) has connections to the True value. Now (simplified) it is enough to look what object is at the end of the connection that runs out from the place of the `True` value where the connection from `(9, 2) 3` object ended. In this way we find out that objects #6 and #7 are red.

4.4.1.5 The Development of the Dolphin Inference Machine

Nowadays, the Dolphin Inference Machine (DIM) is at the start of its development. Currently, DIM is equipped only with basic *modus ponens* like inference mechanism and is able to answer question from conversation: "*Every boy is a child. Peter is a boy. Is Peter a child?*"

The Dolphin system is able to parse and store the `synt` output in the form of TIL constructions, to check the consistence of the database and to answer simple questions. The final aim of the Dolphin development is the full support of working with possible worlds and time moments and the provision of a complex inference tool.

4.5 Experiment of Using TIL in a Simulation System Easel

Easel [Fis99] is a simulation system designed for gathering, processing and display of information about various active entities of the simulated world, about their interactions, and about their collective global effects.

For the description of the entities in the simulated processes, Easel uses a property-based type system. In the system a *type* provides a (complete) description of a template of an object. Any entity is then an *instantiation* of such template. The type can also be thought of as a class of all objects that comply with the set of properties stated in the type template.

Together with the inheritance of type properties and arrangement of types in one hierarchy (with the root type *all*), Easel language resembles common object-oriented (OO) programing languages and formalism. However, as an extension to the usual OO classes, Easel emphasizes the need for type manipulations with symbolic techniques. The definitions of subtypes are described with adjective modifiers similarly as in natural language expressions:

```
    example(): action is
        fruit: type;
        round(any): type;
        green(any): type;
        apple: type is round fruit;
        golden_delicious: type is green apple;
        my_apple: golden_delicious;

        confirm my_apple isa fruit;
    example();
```

4.5.1 Using TIL and Easel in Applications

The specifications of Easel types is suitable for limited simulations with strictly specified bounds of the simulated world. However, in case of complicated descriptions of such extent that reaches the complexness of the real-world situations, the descriptiveness of the types in Easel would run against difficulties due to their underspecification.

The possible combination of both, TIL types and Easel types, in one application seems to be a promising completion of the type specification techniques. TIL allows to place no limits to the expressiveness of the descriptive language used to describe the types and subtypes used in the simulation and (possibly) can make use of any properties from the intensional base of natural language.

The confirmative claims that are currently allowed in the Easel language, would not need to be closed to browsing the type hierarchy, but, with the use of the inference mechanism in TIL, it could analyze any consistent proposition about the type. The above stated example could look as follows:

```
example():  action is
    # the primary properties
    Fruit is a class of individuals.
    To be round is a property of individuals.
    To be green is a property of individuals.
    # the definitions and instantiations
    Every apple is a round fruit.
    Golden Delicious is a green apple.
    The variety of my apple is Golden Delicious;
    # statements
    If I eat a fruit, I am healthy.

    # claims to be confirmed
    confirm that my_apple is a fruit;
    confirm that if I eat my_apple, I am healthy;
example();
```

The form of the description of the situation, i.e. the type definitions, can be written in natural language statements about the discussed elements and their properties. An automatic parsing system can translate the specifications into the following objects and propositions:

`example(): action is`

# the primary properties

$\textbf{fruit} \ldots (o\iota)_{\tau\omega}$

$\textbf{round} \ldots (o\iota)_{\tau\omega}$

$\textbf{green} \ldots (o\iota)_{\tau\omega}$

# the definitions and instantiations

$(\forall t)(\forall x)\textbf{apple}_{\mathsf{w}_{\mathrm{TIM}}t}x \supset (\textbf{fruit}_{\mathsf{w}_{\mathrm{TIM}}t}x \wedge \textbf{round}_{\mathsf{w}_{\mathrm{TIM}}t}x)$

$(\forall t)(\forall x)^0(\mathsf{Golden\ Delicious})_{\mathsf{w}_{\mathrm{TIM}}t}x \supset (\textbf{apple}_{\mathsf{w}_{\mathrm{TIM}}t}x \wedge \textbf{green}_{\mathsf{w}_{\mathrm{TIM}}t}x)$

$(\exists my\_apple)^0(\mathsf{Golden\ Delicious})_{\mathsf{w}_{\mathrm{TIM}}\mathsf{t}_0}my\_apple$

# statements

$$(\forall t)(\forall x)\Big[\big(\textbf{fruit}_{\mathsf{w}_{\mathrm{TIM}}t}x \wedge \textbf{Does}_{\mathsf{w}_{\mathrm{TIM}}t}I\ [\textbf{Perf}_{\mathsf{w}_{\mathrm{TIM}}}[\textbf{eat}\ x]_{\mathsf{w}_{\mathrm{TIM}}}]\big) \supset \supset \textbf{healthy}_{\mathsf{w}_{\mathrm{TIM}}t}I\Big]$$

# claims to be confirmed

`confirm` (*find a match*)...

$x : \textbf{fruit}_{\mathsf{w}_{\mathrm{TIM}}\mathsf{t}_0}my\_apple$

`confirm` (*find a match*)...

$x : \textbf{Does}_{\mathsf{w}_{\mathrm{TIM}}\mathsf{t}_0}I\ [\textbf{Perf}_{\mathsf{w}_{\mathrm{TIM}}}[\textbf{eat}\ my\_apple]_{\mathsf{w}_{\mathrm{TIM}}}]) \supset \textbf{healthy}_{\mathsf{w}_{\mathrm{TIM}}\mathsf{t}_0}I$

`example();`

In the case of argumentations about instances of a type, like *my_apple* here, we use the current reference time t_0 and the actual reference world $\mathsf{w}_{\mathrm{TIM}}$, which are related to the working environment of the processing system.

The mechanism of analyzing questions and inferring derivations from other facts is described in more detail e.g. in [Tic82, Hor02]. Finding a *match* means a request to infer (from the underlying knowledge base) the value x of the specified TIL type, i.e. the truth-value o of the extensified proposition on the right hand side of the match, here.

Even if the above stated examples do not use more power, than a temporal first order logic system would provide, the TIL theory ensures correct analysis of much more complicated expressions, like *intensional roles* or *belief sentences*.

A straightforward reuse of the TIL analysis and TIL inference system is, however, not so easy as a simple incorporation of some other, fully described algorithm. TIL analysis needs to work with something as complex as the natural language and the ambitions of TIL do not allow to make simplifications of the matter.

4.6 Long Way to Full Natural Language Semantics

In this chapter, we have introduced the main ideas and basic structures of Transparent Intensional Logic. We have also argued for the TIL constructions being the meaning bearer for any natural language expressions. The part of the Normal Translation Algorithm for logical analysis of a natural language sentence that is responsible for building a single clause construction was explicated here. We have concentrated on the description of the lexicons needed for analysis of the lexical items and for analysis of particular verb frames. Several examples of existing lexicon entries have been displayed. We have also proposed and specified a way of combining the descriptions of lexical entries in the dictionary with the respective information obtained from lexical semantic resource, namely WordNet.

The text presented a first outline of comparison and integration of the two approaches, namely logical and linguistic, to the semantics of verbs in a natural language. We are aware that this work is still in a great progress and the results so presented are preliminary. Still, we are convinced that the research task we aim at is a relevant contribution to the semantics of natural language.

We have shown that pursuing such a research is reasonable and comes up with a new viewpoint to the meaning of verbs. In this way we extend our knowledge in the important way. Actually, we are dealing with two deep levels of the meaning description and a question may be asked which one is deeper and better. Our answer is, do not contrast the two levels, and make use of both of them. In this way we believe to integrate them into one compact whole and perhaps obtain a unique data structure. The results of the presented research can be immediately applied in the area of knowledge representation and in the long-term Normal Translation System project that is being prepared. We have not tackled the other deep descriptions, such as the method that exploits the tectogrammatical level as it is presently applied in PDT [Haj04a]. This, obviously, is a topic of another text.

We have briefly presented basic features of the Dolphin system for handling structured knowledge of TIL with the question answering capabilities.

As a complement to the extended type system of TIL, we have presented an experiment with the Easel language with its property-based types. We have formulated the assets of using the two approaches in one application and showed the need for working implementations of TIL natural language analyzer and TIL inference system based on the work of TIL's author and other researches.

Chapter 5

Application in Dialogues – the Electrical Power Systems Simulation

Emergent systems and intelligent agents are in the focus of intense attention in many areas of computer science and artificial intelligence. Their main advantage can be seen in the automated adaptation to changing environment and in maintainable control of such system behaviour. There is no need to specify global constraints for the system as a whole. The high degree of autonomous behaviour means that the user can concentrate on the most relevant areas.

Our research focuses on a future architecture of electrical power system simulations able to cope with the changing conditions. The electrical power industry in the Czech Republic and all post-communist Central Europe countries changed dramatically in the last decade. The changes on the market and the development of information technology as well as the power sector call for new methods that are able to cope with the evolving environment. There is a need for flexible adaptation of the current power system operations. Optimizations of power networks have to take into account the market strategy. The maintenance and network planning must be in a balance with the return of investment for the distribution companies.

Intelligent agent systems are described in many new artificial-intelligence monographs (see e.g. [RN03]). The application of intelligent agents in power systems control and operation is discussed in [Reh03]. Intelligent autonomous systems are seen as systems that can react intelligently and flexibly on changing operating conditions and demands from the surrounding processes. The analysis of electrical networks from the agent perspective is tackled also in [Aea93].

To enable interconnection of the simulation system with the current power system knowledge bases (e.g. with actual information about the reliability of the used components and current failure data), the systems have to employ standard communication technologies, protocols and models. There are many de facto standards as well as official international standards that the system should be complied with. For example, a communication standards are specified in *IEC 61870: Telecontrol equipment and systems – Part 6: Telecontrol protocols compatible with ISO and ITU-T recommendations* (2002) or *IEC 61850: Communications networks and systems in substations.* Another useful standards for the purposes of data interchange are XML and, especially, web services standards such as SOAP [SOA03] or newly proposed WSRF (Web Service Resource Framework [C^+04]). The interfaces of the developed systems should be compliant with *IEC 61968: System Interfaces for Distribution Management* and *IEC 61970: Energy Management Systems Application Programming Interfaces.*

The fault-tolerance and quality assurance of the power supply is very important and constantly evolving science branch which demands intensive research. Because of the geographic dissipation and big financial cost of the electrical power systems (EPS) facilities maintenance, the required reliability cannot be reached simply by redundancy. It is necessary to search for new low-cost but effective means of the required reliability assurance.

One of the possible approaches to this problem is regular checking of the facility operation. Failure data are gathered in failure databases with a precise description of the failure cause, time, severity and the description of the failed component – its manufacturer, model, serial number etc. This effort allows a statistical analysis of the reliability of the particular facility series, types etc.

This type of the failure database is for the whole Czech republic built and maintained by the research team for creation and categorization of failures records for distribution equipment and outages of supply at all voltage levels in the Technical University of Ostrava in the team of Stanislav Rusek. Utilities for the analysis of the data in this database are also developed in Ostrava in the team of Václav Snášel. The aim of this effort is to develop a new methodology for the power system facility maintenance based on the condition-centered approach rather then the contemporary used preventive time-based approach. Power system facilities are well tested in the process of their design and manufacturing but there's a serious lack of testing in the process of their real operation.

The new methodology should be flexible enough to allow fast and effective control even for facilities that lack measured reliability data. The Technical University of Ostrava with the help of other universities in the Czech Republic has made an interdisciplinary team which is focused on the development of new methodology for the power system facility condition checks, outage prediction, black-out risk reduction and early post-fault recovering.

The team at FI MU Brno is focused on the development and usage of multi-agent systems in power system facility simulation, monitoring and control.

Enterprises running power systems are making big efforts to make the reliability assurance process more flexible and adequate to the real conditions of the power facilities. Simple time-based controls of the facilities do not take into account the rate between the maintenance costs and the money loss caused by the failures in the energy delivering process.

A more appropriate way of dealing with the reliability assurance will be the new methodology of maintenance based on the actual state of the facility and the amount of the energy delivered (or "potentially undelivered") by the facility.

To achieve more reliable power delivering many of the facilities are backuped. Routine action after failure is to change the active lines topology or to switch to a backup facility to restore power delivering. Each failure is precisely recorded in a failure database (SCADA, Supervisory Control And Data Acquisition [SB95]). The process of the power system maintenance and management, failure recognition and function restoration can be optimized at least from the economic point of view. The problem is somewhat classical: to minimize the overall costs (maintenance, restoration, undelivered energy) which is a function of the facilities condition and its features, dynamic network topology and also the restoration process methodology. The problem is that the function is not computable in a straightforward way and it is not even clear which variables are to be measured to figure out the function.

Our work has two main motivations: (1) to support the development of the condition-oriented facility maintenance methodology by the information technologies, and (2) to develop a software for simulation of the various features of the power system facilities. The task is concentrated mostly on simulation of the electrical energy flows, outages due to facility failures, and their impact on the overall sum of the non-delivered energy.

In its first stage, the developed system (called Rice) is meant to be a research tool extending classical tools for reliability assurance like SCADA databases and statistical tools for failure-data exploration. There is a narrow connection between the exploration of the data in SCADA databases and the developed simulator.

The simulation of power networks represents an appeal to the research thanks to its complexity and also to the potential positive economic aspects of the outage prevention [Ami01]. The ability to compare the results of potential situations leading to a network outage and even the ability to automatically compute the economic loss of the engaged power consumers are the most welcomed features of the Rice system described in this chapter.

Rice is a prototype of a software system designed for simulation of the energy flow and the errors and outages propagation in the power networks developed at the MU NLP Centre by Miroslav Prýmek under the supervision of Aleš Horák. The architecture of Rice is based on the principles of the multi-agent software design [Jen00]. The aim of the work is to develop a software tool which will be able to simulate complicated behaviour of the power systems in the dynamically changing environment.

The basic idea of the system design, according to the multi-agent approach, lies in modelling the whole system "from the bottom upwards." Definition of a primitive component behaviour and relations and communication between them is the main principle of the definition of the whole system characteristics.

Parts of the project work described in this chapter have been published in [SPHS05, PH05a, SHKC05, PH05b, HP06c, PH06b, HP06b].

5.1 Multi-Agent Framework for EPS Simulation and Monitoring

The first phase of analysis and design of the framework for description and control of electrical power networks processes was inspired by artificial life (AL) systems. We have taken the most influencing features of biological systems, which are difficult for simulation by AL methods. The following section deals with the criteria, but applied to the simulation of electric power networks.

Topology Electric power networks (EPN) topology can be multi-path, i.e. in the oriented graph representing an EPN, there could be several different paths from node A to node B. But on the other hand, the topology is quite stable and defined precisely.

As opposed to the biological systems, EPN topology is simpler and hence easier to simulate – it has no features of randomness, dynamics or inaccuracy, which cause the most problems in modelling a living system.

Interaction EPN behaviour is limited to local interactions. This fact enables solving the problem of simulation by means of the network of many locally interacting agents, which do not depend on inner states of each other and act according to their own information and intentions.

Emergence/genesis Like in biological systems, the complexity of EPN behaviour (and, hence, the difficulty of its prediction) is expected to depend on the complexity of the system's structure, and parallelism and independence of its elements.

The behaviour of particular basic elements of EPN is predictable, which makes the simulation straightforward and highly plausible. If the element's behaviour can be predicted analytically, we can apply the "behavioural" approach (see [Ski57]), i.e. we treat an element as a black box, observe and fix its explicit reactions on various stimuli, and reproduce the same behaviour with the simulator.

Here we should clearly state two hypotheses, which compose the basis of our work:

1. the behaviour of the separate EPN elements is easy to determine, define and simulate.
2. if we unite separate elements into a network and supply the sufficient support for their parallel, fast (not only real time) and adequate connectedness, the whole system will appear, whose complex behaviour is comparable to the real EPN.

Evolution The goal of an AL simulator is to create a system with "open-end evolution" [Ray07], while the goal of the EPN simulation is to approximate the real functioning of a real EPN. In this case, we are not interested in the evolution of the whole system; rather we focus on the approximation of elements behaviour in reality.

The law of evolution in the form of the evolution algorithm is useful to the EPN simulation, but with different goal and in different format. Instead of the competitiveness between separate elements ("organisms") we must apply the optimum searching algorithm.

Fitness Function As could be seen from the previous paragraph, in case of EPN the fitness function plays a role, which differs from that in AL. For our purposes, we define the fitness function in such a way, that it reflects the extent of the difference between the simulation and the real element. An advantage can be seen in the fact, that the fitness function of particular elements of EPN is easily measurable, as opposed to biological systems, where we often face the "hidden" fitness.

The arguments stated above clearly show that methods and models, applied to biological system simulation, are at least to the same extent effective for the EPN simulation. This conclusion is based on the fact that, in case of EPN, the criteria demonstrate lower complexity, sometimes absence of the elements or features, which cause the most problems with the predictability and effectiveness of the simulation.

For the implementation of the methods, we have decided in the first phase to adapt the Kairos system [Prý04]. This tool is modular and is based on rules of object-oriented programming. Primarily, it was designed for biological systems simulation. Solving the problem of EPN modelling, some features of Kairos were modified significantly, as to adapt to the new goal of research. Kairos is designed for Unix, but it can be ported to other platforms. Three independent blocks compose the original version of Kairos, which interact through the TCP/IP; only two of them are applied for EPN simulation – the functional core and the visualization tool. Some modifications that were made to adapt Kairos to the goals of EPN modelling are described in the following paragraphs.

Network nodes The network nodes are static according to the topology and that is why we identify them with life environment fields in Kairos. Herewith, some features that are usually unique to organisms must be used – adaptability, communication, complex behaviour etc. The network nodes are therefore implemented by synthesis of features for fields and organisms. The leading wire is also a network element for the sake of error simulation, of course.

Topology The Kairos system has no limitations to the topology of the simulated environment, since Kairos is based entirely on the local interactions between environment fields. We exploit this feature and, instead of two dimensional rectangular grid that is usual in the life system simulators, we use a structured network appropriate for modelling real electric power networks.

Used primitives The Kairos system provides several specialized primitives, out of which we use mostly the current and energy transformation primitives. The only change that is needed for the sake of effectiveness is the definition of energy as a relative value. With this, we do not need to model a constant energy current with a constant data flow, but only with a single information about the starting values of the current.

Visualization The visualization subsystem for EPS processes has been completely redesigned, the energy flows visualization techniques do not share many common features with artificial life simulations. Here, the subsystem design is based on existing concepts and implementations stated in [Reh03].

5.1.1 The Rice System Architecture

Further development of the EPS simulation evolved in a complete new system called *Rice*. The Rice software framework is an experimental implementation of a power system simulator, which implements all of the above-described features needed for developing the new reliability assurance methodology. It is based on these main principles, described further: (1) it is a multi-agent system, (2) it can be used to study behaviour of the dynamically changing environment, (3) as to practical implementation, it is decentralized, modular and open. This is necessary especially for use in geographically dislocated areas (e.g. online facilities monitoring).

The design of the Rice multi-agent framework is based on the following basic requirements:

Decentralization Because of the power systems nature (especially their geographic dissipation), the system allows decentralized distributed operation on many computers interconnected with the standard type of computer network (LAN, WAN, Internet). This requirement is even more important for the process of real-time on-site monitoring of the facilities.

Platform independence The application environment is very heterogeneous. It is absolutely necessary to use technologies independent on hardware platform and operating system. The framework should also support wide range of programming languages.

Performance scaling Hardware requirements of the simulation and the range of future applications is hardly predictable. The framework therefore supports easy performance scaling by extension of hardware and software dedicated to it. This extension should not require large changes to the existent architecture and implementation.

Modularity and extensibility The framework is flexible, open and extensible for addition of new features and modules which appear to be needed during the process of dissemination and testing. In such case, changes to an existing code should be as minimal as possible.

Open standards We must take into account a possible future need to integrate the proposed system with the contemporary software and hardware solutions of the power system facility vendors. To make this integration easier, the system is based on standard technologies – with open specifications and implementations.

Security Communication between system modules usually goes through untrusted channels. Privacy and authenticity of the particular communication acts must be still guaranteed on the level common in the industry solutions.

Low-cost practical application The framework is to be based on commonly used technologies. Only then we can assume an easy and low-cost practical application in the production environment. If any usage of non-common technologies will prove to be necessary, it should not appear in many parts of the system, especially not in the user-oriented parts of the system.

After the specification of all the system requirements, the decision was made to implement the system as a multi-agent system based on the standard technologies commonly used in this area – a combination of two communication protocols, CORBA and KQML. CORBA standard has freely available specification and there are many implementations – proprietary ones (e.g. Visibroker, PeerLogic, IONA) and even free ones (e.g. OmniOrb, Orbit, Mico). In our prototype implementation, we use OmniOrb (in Python) and Sun ORB which is part of the Java SDK. An indispensable advantage of the CORBA standard is that all ORB implementations conforming to the specification would integrate well within the system.

Usage of the CORBA-KQML solution was presented in several projects, e.g. [DTH99], [COB04], [BSS98]. The multi-agent approach has been used even in the area of the power systems monitoring and control (e.g. [NS04], [MMH03], [CVJ92]). The Rice implementation is, however, the first system that combines these approaches with processing of SCADA databases and with modelling of economic loss with energy not-supply.

As discussed in [MMH03], old "hardwired" solutions of the power systems monitoring have many problems caused by their limited flexibility which we are going to overcome by using the open and extensible multi-agent system implementation. This approach will be strong enough to satisfy all our requirements.

5.1.1.1 The Multi-agent Approach

Rice is based on the principles of the multi-agent approach (see the Figure 5.1). Each facility of the power system is supposed to be autonomous

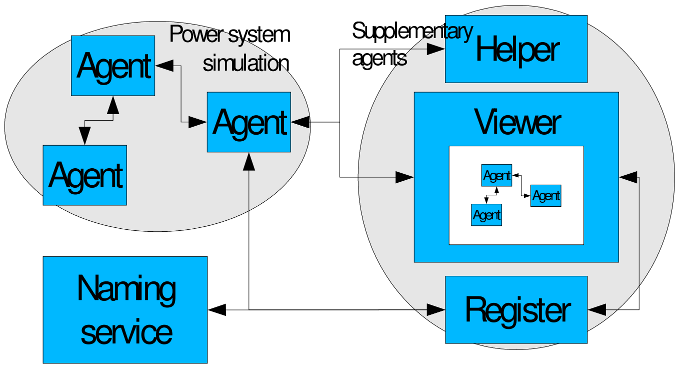


Figure 5.1: Role of supplementary agents in the Rice system

(self-deciding and autonomously reacting to system events) and so represented by one software agent.

A general description of an agent-based software engineering methods, that form the basis of the presented system, can be found e.g. in [Jen00]. Case studies of actual applications with agent-based architecture in the industrial control systems are presented in [JB03].

Key idea of the multi-agent system is the idea of the *agent* itself. There is plenty of definitions what agent is or should be, we can emphasize some common features of agents:

- agent is **cognitive**. It watches its surroundings and makes its own "cognitive map" of the world. All events taking place in the agent's environment are filtered by the agent according to their importance for agent's *aim* and important ones are stored in the agent's *knowledge base* (memory).

- agent is **autonomous**. There is no prepared plan of actions agent should do to achieve his aim. In every time slice agent itself chooses the best plan according to his knowledge base.

- agent is **communicative**. It can ask any other agent for any information he needs to fill his knowledge base. Communication patterns (i.e. topology of the communication net) are strictly *dynamic*. The structure is built up ad-hoc according to agents' needs for some knowledge. There are no "hardwired" communication paths.

- agent is **social**. Main features of multi-agent systems are functions of the *system as a whole*, not any particular agent.

In the Rice system, every power system facility is represented by one software agent. Agents interaction standard is defined by the set of messages agent must understand. These message sets are hierarchically arranged so they grow up into a tree of agent types.

Every agent is therefore viewed as a black box with specified inputs and outputs. To fulfill the idea of multi-agent system, every agent decides on his state only from the state of his inputs. This leads to the *local-interactions*-only system. And further: no information passing between agents is obligatory. Every information must be obtained by the question passed to the particular agent (one question can lead to a time-based or event-based series of answers). This rule asserts that there is no superfluous flow of information.

The "1:1" design (one facility = one agent) of the multi-agent system is necessary for interaction with SCADA databases, as we will see further.

It is not easy to track events in dynamically changing environment. Because we are in the situation where we do not know precisely what variables are of interest for exploring the facilities failure rates (and thus also for maintenance necessity and cost), things are even more difficult. One possible approach is to experiment with the auto-adaptive networks.

The topology and information flow within the system is not fixed. The leading wire agent can either copy the incoming energy flow from its input to the output or it can react to *demands* from the respective power network parts. The network is also dynamic as to the topology. The topology is built up by series of question and answer couples. When a connection between two agents is no more needed (e.g. because the power line connecting them is down), communication is shut down and if there is another agent which can facilitate the same function (and so restore the energy distribution), new connection with it is established. The result is that the system can be used for simulating ever-changing emergent environment and for measuring the efficiency of the energy supply in such an environment.

The application environment is very heterogeneous. It is absolutely necessary to use technologies independent on hardware platform and opa)

```
def rLAgentStatus(self,status):
  if self.msg.inReplyTo=="inputStatus":
    print 'Input status is: "\%s"'\%status
    if (status=="error"):
      print "My input is in error state - switching to UPS"
```

b)

```
def rLAgentStatus(self,status):
  if self.msg.inReplyTo=="inputStatus":
    print 'Input status is: "\%s"'\%status
    self.setRLStatus(status)
```

Figure 5.2: Reactions to zero voltage on the input line

erating system. The framework also supports wide range of programming languages (due to CORBA). In the contemporary Rice prototype Python and Java are used.

The main principle of the Rice system is that each agent can be implemented by a stand-alone process or even stand-alone computer in a computer network. If the communication flow between particular agents is too large, it is possible to implement two or more agents within one process and thus lower the communication overhead.

All agents in the system have the capability to communicate through the CORBA and KQML protocols (see the Section 5.1.1.2). Each agent has a defined type (denoting a set of messages which are accepted and understood by the agent) and it is identified within the network by its distinct name and a given identification number. The agent types are implemented as a hierarchy in which each level is assigned a set of mandatory KQML messages which every agent of this type must understand and must be able to respond to it. The hierarchy looks like that:

- Agent
  - Real-life agent (represents a particular power system facility)
    - Line
    - Facility
      - Source

        · Transformation station
        · Switching station
        · Consumer
    – Organizational and auxiliary agents
        * Register
        * Helper
        * Viewer

Each agent that belongs to some category must be able to respond to every message from the defined set of messages. For example, agent of the type `Source` must be able to respond to all messages mandatory for types `Source`, `Facility`, `Real-life agent` and `Agent`.

Register The Register agent holds a database of all agents in the system, their name, type and Agent ID (AID). Every agent must notify the register about its existence before joining the network. The register then passes a communication key to the agent. This approval for the network joining serves as the certificate of authenticity and can be validated by each agent in the network. The key also plays a role of an encryption key for securing inter-agent communication. Agent is also given an AID. The register agent can return an agent network address as the response to a query based on agent AID, name or type.

Helper The Helper agent implements all global functions that are not bound to any particular agent but are relevant to the system as a whole – e.g. system identification values, supported protocol versions, description of the allowed agent types, global configuration constants etc.

Viewer The Viewer agent performs a graphical display of the events that take place in the system (see the Figure 5.3). There can be plenty of viewer agents and each of them can visualize the state of the system in a different manner – e.g. to plot a graph or to save the data in a database. Viewer agent connection to the network is performed strictly dynamically – after joining, the Viewer asks the Register for notifying about network addresses of all viewable agents (KQML performative `subscribe`) and then asks each of the agents to constantly provide information needed for visualization of the agent (the same performative).

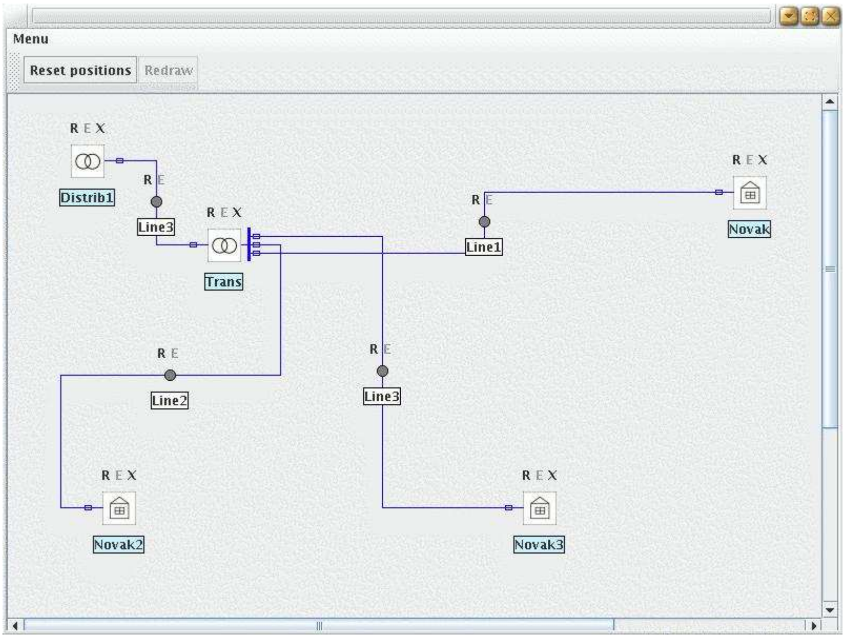


Figure 5.3: Example screen of the Viewer-type agent

Real-life agents In the sense of multi-agent systems theory, each part of the system is an agent. By the term "real-life agent" (RLA), we mean an agent representing a particular power system facility (it has an equivalent in the real life). RLAs have (as opposed to other agents) a huge set of mandatory messages regarding the power flow description.

Line A power line is the only part of the network which builds up the network topology (i.e. its interconnection). Each line is labeled with names of the two agents representing its input and output. Other RLAs (of the `facility` type) are passive – they wait for the connection and do not initialize it in any manner. So the topology of the RLAs remains stable in the whole run which corresponds to the reality of the power systems. On the other hand, the connection with other agents is dynamic – it

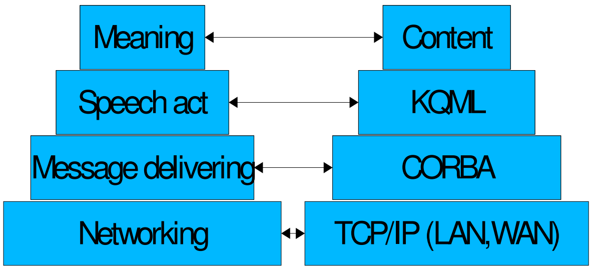


Figure 5.4: Communication layers

is possible to assign new monitoring agents to a running system.

5.1.1.2 Communication between Agents

The basic assumption of the system topology is that agents can be located on separate machines or at least separate processes within one machine. Hence the communication between them must be constituted by some kind of network protocol – in our case it is TCP/IP. But this protocol represents only the lowest layer of the communication which is extended by three other protocols on the top two levels of the OSI model [OSI84] – they are CORBA, KQML and the content language itself.

The first layer, TCP/IP is the standard Internet communication protocol allowing the system to consist of geographically dissipated components. The idea is that some of the simulating agents could be in future replaced by on-line sensors monitoring a real facility operation. This change is seamless and does not demand deep code intervention because of the multi-agent architecture and the KQML flexibility.

The second layer, CORBA (Common Object Request Broker Architecture [COR04]) is a standard architecture for inter-process (and even inter-machine) procedure calling, used widely in system integration solutions and decentralized software. It is strictly object-oriented. The CORBA object programming interface is defined in an abstract Interface Definition Language (IDL [IDL02]) which can be automatically translated into many real programming languages (currently, we use mappings into Java [CJa02] and Python [CPy02]). This language is formally similar to structures used in data modelling meta-language UML (Unified Modelling Language). IDL also includes standard specifications of general data-structure mappings into many programming languages. Thus it is not difficult to integrate code pieces written in different languages and running on different operating systems and machines. All CORBA operations all strictly platform independent – in the sense of hardware and software equipment of the machines.

A big advantage of CORBA is the fact that objects communicate with each other in the same way as when they are in the same process – CORBA compliant ORB (Object Request Broker) facilitates all data transformations and network transportation absolutely transparently. This feature is especially important for possible system rearrangements for the performance enhancement.

Another very important feature of CORBA which is used in our framework is so called CORBA Naming Service (NS). This service is similar to the Internet Domain Name Service, DNS. CORBA NS converts an object name to a distinguished network address. Thus the knowledge of an agent name and an address of the CORBA NS server is sufficient for establishing a connection with the agent. NS server provides an agent address on demand. This way it is possible to form an architecture which can be subject to changes and is absolutely independent on the part of network the agent is physically running. This key feature makes it possible to reach the required system flexibility. Agents need to know only the names of agents they are about to communicate with – all other information is gathered dynamically.

The third layer, KQML (i.e. the Knowledge Query and Manipulation Language [FFMM94]) is a language developed specially for usage in multi-agent systems. It is an abstract definition of the message purpose, not the message content itself. This way, the meaning of the message is transparently separable from the message object, which is very useful for system integration and definition of abstract message types. The detailed message flow schema between two agents is depicted in the Figure 5.5.

The last layer is the message content itself. Thanks to the KQML structure, it can be defined in many languages with arbitrary structure. The message content language used can even vary between individual messages and can be dynamically negotiated by the communicating agents.

For instance, a definition of a data structure and an interface of an agent which is able to communicate in KQML looks in IDL this way:

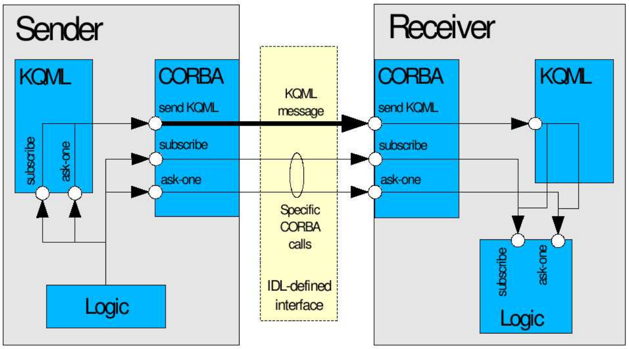


Figure 5.5: The Implementation of the Rice Communication Schema

```
struct KqmlMessageStruct {
   string performative;
   long sender;
   long receiver;
   /* ... */
   any contents;
   long inReplyTo;
   string force;
};

interface KqmlReceiverInterface {
   exception badKQMLMessage{};

   // send KQML message - standalone args
   void kqmlMessageArgs(in string performat, in long sender,
       in long receiver, in any contents, in long inReplyTo,
       in long force)
          raises (badKQMLMessage);

   // send KQML message - struct
   void kqmlMessage(in KqmlMessageStruct message)
          raises (badKQMLMessage);
};
```

For each of the supported languages, there is an IDL compiler which builds a skeleton in the desired language. The skeleton corresponds to the structures defined in IDL (in the above example, it will build the `KqmlReceiverInterface` in the Java language).

The KQML layer is built above the CORBA layer which facilitates the (virtualized) connection between agents. The KQML language is based on the linguistic theory of the Speech acts [Smi90], published by Searle in [Sea69]. One of the basic ideas of this theory is that every communication act can be categorized as an announcement, a query, a demand etc.

The KQML communication is strictly divided into two levels – level of the speech act resolution and the message content resolution. For each speech act type there is one or more so called "KQML performatives." Performatives provide basic information about what type of information or action an agent demands.

The KQML language is defined as an abstract query and manipulation language. The basic concept is that each of the communicating agents has its own knowledge database concentrating knowledge about the outer environment and the agent itself. But this is only a formal concept. In reality the agent can work even strictly reflexively – but it must represent its actions as manipulations with the knowledge database. For this reason we call the desired data structure on which the KQML operates a *Virtual Knowledge Base* (VKB).

KQML has two fundamental concepts: (1) each agent is autonomous and only it decides what to do in a particular time, (2) agent A can ask agent B for information from its VKB. KQML contains a set of performatives which express a desire of an agent A for agent B to make an effort to achieve some state of the environment.

The KQML message envelope defines from whom, to whom and when was the message delivered and in which form the sender expects an answer. The content itself is communicated in the content language, which can be any possible language which is adequate for the message content expressing and which is understood by both the agents. In practice, usually Prolog or KIF[1] are used.

For an illustration of the inter-agent communication we take a typical situation in a power system simulator (the example is simplified for better

[1]Knowledge Interchange Format [GF92] was designed especially for this purpose.

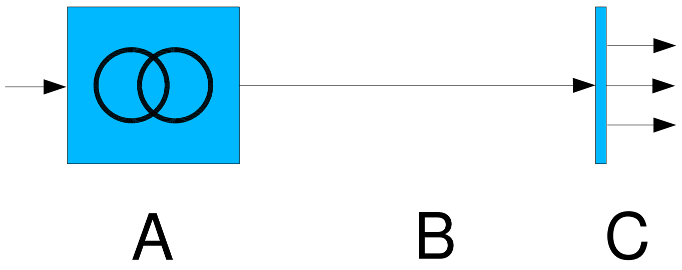


Figure 5.6: Example of EPS agents' communication

clearness): agent A represents a transformation station transforming a very high voltage to a high voltage. Agent A is connected to agent B which represents high voltage line and which is connected to agent C representing a distribution point (see the Figure 5.6).

Agent B asks agent A for a notification of every A's state change and every A's output voltage change:

```
(subscribe
   :sender      B
   :receiver    A
   :timestamp   1113340454
   :reply-with  query_1
   :language    KQML
   :ontology    KQML_ontology
   :content     (ask
        :sender B
        :receiver A
        :in-reply-to query_1
        :reply-with 2
        :language Prolog
        :ontology Power_system
        :content out_voltage(X), state(Y)
     )
)
```

As soon as the agent forms a KQML message, it contacts the CORBA Naming Service and asks it for a network address of the recipient (in the case that it is not in his VKB yet) and contacts the recipient (again, in the case that it has not done this already before). The sending of the message itself is realized by a CORBA call which is in the formal sense identical to a call of a function in the same process (programming language “native” call).

By this technique a dynamic connection between the agents A and B is established (i.e. the output of the device represented by the agent A is connected to the input of the device represented by the agent B). Similar connection is established between the agents B and C.

If there is an outage on line B and the current supply is canceled, B sends to C a message of this format:

```
(tell
   :sender  B
   :receiver  C
   :timestamp   1113341454
   :in-reply-to query_2
   :reply-with  query_3
   :language  Prolog
   :ontology  Power_system
   :content     out_voltage(0), state(fatal_failure)
)
```

C reacts to this message in this way: it sets the voltage of all its outputs to 0 V too or switches to another (backup) input line.

The big advantage of KQML is that the agent B can send a message with the same format to the visualization agent or to an agent which loads data into the failure database. It is not necessary to develop a specialized protocol for each communication channel.

It is also possible to change the parameters of the communication with only a little change of the message content. For instance, in the process of building a comprehensive database of power supply failures by gathering information from the agents, the central database requires more detailed data – so the corresponding agent’s `subscribe` message will contain other queries – the `content` field will in this case look like this:

```
:content state(fatal_failure), facility(U), type(V),
         serial_num(W), owner(X), locality(Y),
        description(Z)
```

The type of the speech act has not changed so even the KQML envelope remains the same. Only the message content is different.

An important question, of course, is whether the described four-level communication framework will have enough bandwidth for the power system simulation. A quality agent's design should be based on the following principle: "Agents are communicating only those data which they really need" – i.e. as much data as possible is processed locally by the agent. The other agents should ask only for results of this processing. E.g. the running temperature of a facility can significantly affect the reliability of the facility but no agent should ask for the temperature to deduce the facility's state – instead it should ask for the conditions of the facility or the probability of a failure in the following time period. The amount of the communicated data can be significantly reduced in this way.

The content language should be a language which is a "native" language of an agent. It is very inefficient to choose a language which demands a conversion of existing data to another format and back just for the sake of the communication.[2]

5.2 Human-machine Dialogues with the Rice System

For thousands of years natural language (NL) dialogue has been the most traditional way of communication with humans. Therefore, following this tradition, NL dialogue forms also a new and efficient way of communication between human and computer. NL interfaces provide successful systems for help systems, secretariat systems or tool for complex systems analysis. The purpose of these systems is to handle restricted and well-known domains. In these domains, the designers can express the rules of understanding user's requests and generating system's responses.

A real electrical power network represents a complicated system of large amount of cooperating appliances. The analysis and prediction of failures in such systems (see [KMG01, PSH01]) is a difficult task that often suffers with overspecification problems. The aim of our research

[2]here we speak of communication between agents, not human-machine communication.

is to explore the possibility of enhancing the communication with such complex systems analysis by means of natural language dialogue interface.

Methods of automatic analysis of domain-specific texts heavily exploit language resources which are adapted to the respective domain. The creation of the first phase of a basic resource for analysis of dialogues about electrical power networks was published in [PH05a]. The size of the textual corpus described there was about 100 000 token, the morphological tagging of the corpus was ambiguous and the coverage of the syntactic analysis on the corpus was about 50 %. In this text, we show the progress in creating a corpus which is 10 times bigger, does not contain ambiguous morphological tags and the quality of the relevant syntactic analysis has increased up to 76 %.

5.2.1 Building a Specialized Corpus

In this section, we describe the procedure of collecting the domain-specific texts.

As a starting point, we used a set of 14 documents from the application domain – reports, program documentation and technical specifications of electrical devices. These texts together contain approximately 100,000 positions [PH05a]. These texts served for providing a basis for the WebBootCat [BKPR06] tool[3] for extracting keywords and searching the web pages to find and download similar documents.

As a result, we obtained a corpus of about 1 million tokens (words and punctuation). The newly extracted key terms from this corpus contained terms from the electrical power networks domain (e.g. "elektrické zařízení" – electrical device, "elektrická energie" – electrical energy, "napájení" – supply, "proud" – current) as well as more general words such as "material", "producer" or "system" (see the Table 5.1). The key terms search worked also as a check that the system downloaded just the domain-specific documents. A detailed description of the corpus can be seen in the Table 5.2.

[3]developed also in the MU NLP Centre

Table 5.1: Most frequent keywords extracted from the final corpus

word	frequency
zařízení (device)	5 983
musí (must)	3 797
napětí (voltage)	2 919
proud (current)	1 945
elektrické (electrical)	1 859
energie (energie)	1 813
soustava (system)	1 584
měření (measure)	1 484
vedení (wiring)	1 481
části (parts)	1 011
těchto (these)	1 027
provoz (operation)	974
použití (usage)	809
zdroj (source)	759

Table 5.2: Characteristics of the electrical power networks domain corpus

Number of tokens	1 034 511
Number of sentences	43 738
Number of documents	160
Median number of words in a sentence	15
Average number of words in a sentence	24

5.2.2 Morphological Tagging

To be able to analyse the corpus sentences with advanced language processing tools, we need to supplement the corpus by lemmatization and morphological tagging. For this task, we have used the Czech morphemic analyzer `ajka` [Sed05] that provides lemmas and all possible morphological tags for more than 6 million Czech word forms. For the tag disambiguation, we have used a statistical disambiguation tool [Š04] that is being developed at the MU NLP Centre. These two tools together form a Czech morphological tagger.

The overall morphological tagging coverage on the electrical power networks domain corpus was 95.84 %. The most frequent expressions that were not recognized are the domain-specific words (e.g. "rezistance" – resistance), abbreviations (e.g. "VN" – stands for "vysoké napětí", high voltage) and physical units ("kVA", or "kPa"). For successful processing of natural language texts the mentioned morphological analysis tools need to be supplemented with these terms.

5.2.3 Syntactic Analysis of the Domain Texts

Since the syntax analysis is one of the basic steps in natural language understanding, we plan to use the `synt` parser[4] as a part of the developed dialogue interface. Therefore, it is important to know, how successful is the parser on the texts from the particular domain and what specific phenomena need to be added to the system for good-class analysis of these texts.

In the following experiment we investigated the parser coverage in case of domain-specific texts.

Within the experiment, the whole corpus was split into sentences, using the default sentence boundaries provided by the WebBootCat system. Then, the `synt` system was executed on these sentences and the number of accepted sentences was computed. As can be seen in the Table 5.2, the total number of sentences in the corpus is nearly 44 000. The results of the experiment can be seen in the Table 5.3. In the Figure 5.7, there is an example of a `synt` derivation tree for one of the corpus sentences.

The Table 5.3 shows that the parser has accepted about 62 % of the corpus sentences, which is much less than in the case of common sentences

[4]see the Chapter 3

Table 5.3: Results of syntactic analysis with the `synt` system

Sentences accepted	27 209
Sentences rejected	16 529
Parser coverage	62.2 %
Average parsing time	1.96 s

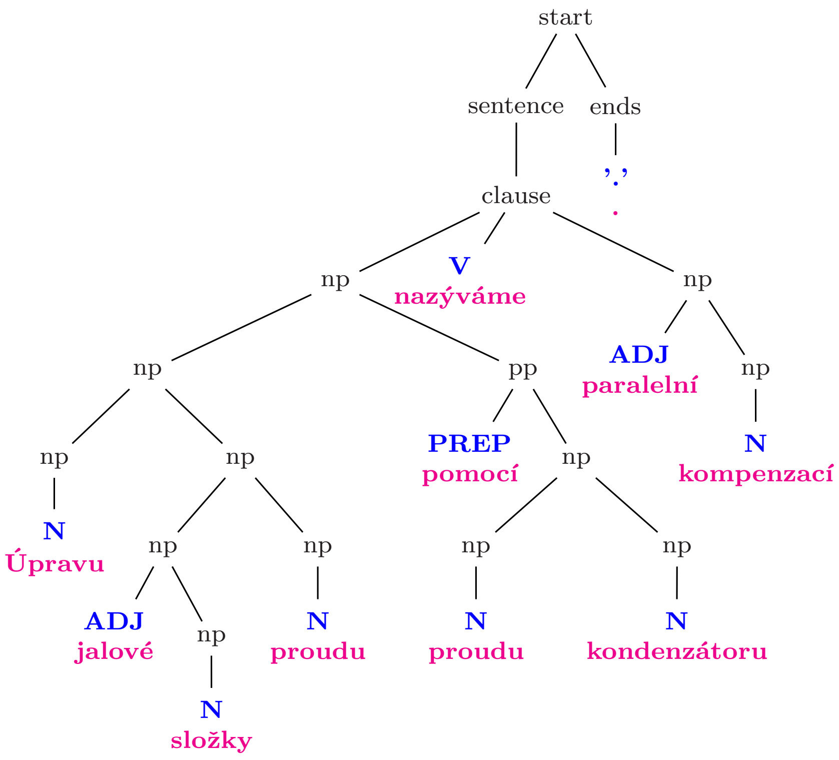


Figure 5.7: Example of a derivation tree for one of the corpus sentences.

occurring in a reference corpus. It can also be seen that the average parsing time is significantly higher than the parsing time measured on the reference corpus. One of the reasons for this fact is the high average sentence length (see the Table 5.2).

The main reasons for the low coverage of syntactic analysis of the corpus are as follows:

- the frequency of yet unknown key terms is higher in the domain-specific texts
- special expressive techniques of the technical texts (frequent use of abbreviations, table and figure references, numbers, equations, . . .)
- possible errors in the sentence-splitting mechanism in the WebBootCat system

An example of difficulties described in the first two points can be illustrated e.g. with the sentence: "Celkový instalovaný příkon cca 6 x 400 = 2400 kW." (Total energy input installed (is) cca 6 x 400 = 2400 kW).

In order to verify the last of the previous three points, we have made an additional experiment. We found out that the corpus contains (among others) many very long sentences (up to 1 000 words), so it is evident that some sentences are marked badly. Therefore, we filtered out the sentences with a high number of words and measured the parser coverage again. The results are shown in the Table 5.4. At shorter sentences (that are more probable to be marked correctly), the parser coverage is significantly higher, up to 76.4 %. These numbers show that some more fine-grained sentence splitting procedures (than the default one that is integrated in the WebBootCat system) should be applied to this kind of corpus.

5.2.4 Designing an Intelligent Dialogue Interface

Dialogue interfaces are interfaces that use natural language for communication with users. Designing an intelligent dialogue interface is a complex task that covers several fields in the area of human language technology, including morphological, syntactic and semantic analysis, information retrieval, discourse analysis and language generation. Designing such complex system with full functionality for general use is unfeasible with the current level of technologies. Nowadays, such systems are always *domain dependent*, i.e. tailored for a particular domain.

Table 5.4: Parser coverage measured on the sentences with limited length

sentence length limit (words)	number of sentences	parser coverage (percent)
None (all sentences)	43 738	62.2
40	36 043	68.5
30	32 219	72.0
20	24 430	76.4

At the first stage, we have focused on development of a question answering system for searching in large databases of complex facts called *UIO*. Currently the system is able to process natural language queries over the administration system of the Masaryk University – it contains data about 35 000 students, their studies, courses and other information. We are now adjusting the system for queries over the database of electrical network failures. The UIO system contains a general algorithm for question analysis, information retrieval and answer generation and it is being equipped with knowledge about the domain of electrical network failures, nowadays.

5.2.4.1 The Question-Answering System UIO

UIO is a multi-domain oriented question-answering system for the Czech language incorporating advanced natural language processing techniques, designed and developed by Lukáš Svoboda in the MU NLP Centre. A simplified scheme of the system is depicted in the Figure 5.8.

The analysis of Czech sentences has to cope with practically free word order and a high degree of ambiguity caused by the inflectional nature of the language. UIO tries to solve these problems by using partial analysis with known question schemata.[5] First, the morphological analysis of the user question is performed, which finds the corresponding lemma(s) and part-of-speech tag(s) of each word. The morphological disambiguation is usually not performed, the question is analysed with all possible

[5]Please note that the description is simplified, detailed information can be found in [SP04]

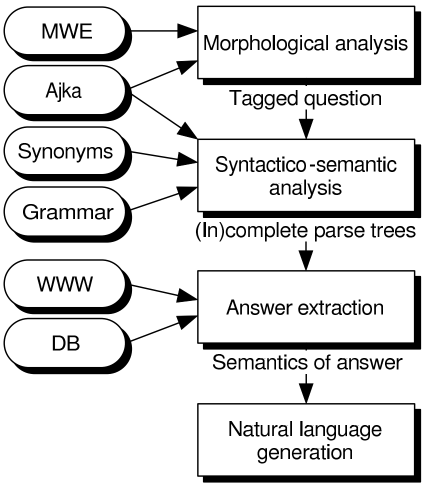


Figure 5.8: *UIO flow chart.*

lemma/tag combinations. Specific part of this phase of the analysis lies in recognition of multi-word expressions (MWE). For UIO, a special MWE module has been created [Svo04]. This module works with a database of MWEs (containing more than 100 000 items) and provides functions for MWE inflected forms and lemmatization.

The result of the morphological analysis is used as the input for the syntactico-semantic analysis of the question. For this purpose, a special semantic-based grammar is used. It is based on a context-free grammar which is augmented with tokens carrying information about the semantics of the matched grammar. In addition to this, the grammar is extended with special features considering the needs of inflectional languages:

a) besides the usual exact match of terminals, matching based on equal lemmas is supported in order to be able to face the problem of various inflected forms.

b) in order to face the free word order problem, matching any permutation of the given tokens is supported.

c) to improve the flexibility, matching of synonyms is supported.

```
{
    domain => 'ElectricalNetworksFail',
    from_date => { month => 1,
                   day => 1,
                   year => 2007 },
    to_date => { month => 1
                 day => 31,
                 year => 2007 },
    short_circuit => true,
    type => 'ListFail',
}
```

Figure 5.9: Semantic representation of the utterance "*Vypiš všechny vzniklé závady v lednu 2007, kde nastal zkrat.*" (Show all breakdowns in January 2007 where short circuit happened.)

UIO uses a bottom-up chart parser that is able to work with the grammar extensions mentioned above. The result of the analysis is a set of parse trees corresponding to various possible ways of parsing the question. A probability of correctness of the analysis is associated with each tree. The semantic representation of the question is constructed from the tokens carrying the semantic information. The semantics is represented as a list of semantic *attribute/value* pairs with possible hierarchical organization. Several special semantic pairs are available for processing of questions containing superlatives (*What is the highest...*). An example of the semantic representation is depicted in the Figure 5.9

The bottom-up chart parser used in UIO is also able to process partial parsing which is of crucial importance since it happens very often that the question does not match the grammar and cannot be fully parsed. In such case, the parser returns a set of parse trees corresponding to the successfully analyzed parts of the question. From the parse trees, partial semantics of the question can be computed. The capabilities of the system can be presented by examples of different questions that the system is currently able to process and answer – see the Figure 5.10.

Based on the semantics of the question and the associated probability of correctness, UIO first determines the domain corresponding to the question. In the case that the question matches the grammar, this problem is trivial since the domain is associated directly with the root rule of the matched phrase. If only partial semantics is available, the most probable domain is inferred from the available information. Sometimes, there are multiple domains corresponding to the same phrase (e.g. a time table inquiry might refer to a bus connection as well as a flight). In such situations, the answer is extracted for all the domains.

As soon as the domain is determined, the semantic representation of the question is passed to an answer extraction algorithm. The algorithm either detects that attributes needed for answering the question are missing, or extracts the answer from the information source predefined for each domain. The information source can be a local database or a web server that provides the required information. The extracted answer is converted into a semantic representation similar to the one used for representing the question. This representation can be further processed – if multiple domains match the question and there is an empty answer for some of them, it can be ruled out, or, in the case of questions containing superlatives, the answer must be computed from the semantic representation. Then, the result is passed to a presentation module that transforms the information into a text or a table that can be presented to the user.

5.3 Rice Usage Scenarios

There are many ways of possible usage of the Rice system but all of them can be summarized to four main usage patterns. Typical usage of the Rice system will consist of several simulation runs with different properties of the simulated environment and examining the impacts of the changes. The Rice usage scenarios can be grouped according to the examined subject and the means of the environment manipulation:

Global Manipulation, Local Effect The most common usage of the system will consist in the comparison of the behaviour and characteristics of two or more power systems consisting of the same basic components but different interconnection between them (analysis of different power system topology impacts).

In the Section 5.4, the results of an experiment with simulation of the economic loss caused by energy not supply are presented.

Jaké byly příčiny nahodilých poruch v roce 2003? (What were the causes of incidental failures in 2003?)

Kdy bylo vyrobeno zařízení, u kterého se vyskytla nahodilá událost 15.1.2003? (When was the device, which was the source of an incidental event on January 15, 2003, produced?)

Jaká je nejčastější příčina události v roce 2003? (What is the most frequent cause of events in 2003?)

Vypiš poruchy, které se staly 16.1.2003 na vodičích (List failures that happened on January 1, 2003 on leading wires.)

Jaké druhy zkratu nastaly v roce 2003 na stožáru? (What kinds of short circuit appeared in 2003 on a pole?)

Jaký druh sítě byl v roce 2002 poškozen nejvíce? (What kind of network was the most damaged in 2002?)

Kdy mělo poruchu zařízení vyrobené společností ESB Brno? (When did a device produced by ESB Brno company fail?)

Jaké je nejporuchovější zařízení vyrobené společností Kablo Kladno? (Which of the devices made by Kablo Kladno has the highest failure rate?)

Kolik distribučních transformačních stanic bylo mimo provoz 16.6.2001? (How many distribution transformation stations were out-of-order on June 6, 2001?)

Jaká zařízení s napětím 400V byla postižena nahodilou událostí? (Which devices with 400V voltage were affected by an incidental event?)

Jaké se staly události na transformátoru napětí? (What events happened on a voltage transformer?)

Vypiš data, druh události a zařízení se záznamy z 16.1.2003. (Show the dates, event types and devices with records from January 16, 2003.)

Kdy a na jakém zařízení došlo k nahodilým událostem způsobeným zásahy cizích osob v roce 2005? (When and on what device did an incidental event caused by foreign persons happen in 2005?)

Jaká je průměrná délka události v roce 2003? (What is an average length of an event in 2003?)

Jaké jsou příčiny událostí, které vedly v roce 2003 k poškození reaktoru? (What are the causes of events leading to a reactor damage in 2003?)

Figure 5.10: Examples of NL questions that can be handled by the current implementation.

Two power systems with slightly different topologies are described there and the corresponding economic impacts of the outages on particular network nodes are compared.

Such situation represents a typical example of the exploration of the local effects of the global system change. We should note here that the topology in Rice is also defined "from bottom up." Every agent has a definition of its neighbours and there is no overall topology definition so any changes of the topology are easy to represent. No intervention into the agent behaviour definitions are needed.

Local Manipulation, Global Effect The second usage pattern is similar to the first one but one more Rice feature has to be used – the data gathering and analysis tools (in the case that the explored global effect is composed of individual local effects).

Rice includes a data gathering mechanism similar to the failure databases used in the real power system monitoring. Each agent has its own data storage called *event database*. All events relevant for the particular exploration are stored in this database in a standard form. The system includes a tool for merging these databases for preparation of overall statistics such as the total outage time, the outage-caused losses or the overall energy consumption.

An example of such an exploration can be also found in the Section 5.4.

Local Manipulation, Local Effect The third usage pattern is used for exploration of the impact of a change in the behaviour of some particular network component on other components. This action cannot be reduced to a relation between two components because the results of the change are often obtained by a non-trivial function over a whole inner-standing network segment.

Tests of this kind are possible in Rice, however, no such test has been realized yet.

Visualization As we have said already, Rice includes tools for exporting events taking place in the network into a database. But this events can also be imported on-line from the databases. With this feature, Rice can be used for visualization of the processes taking place in the real power systems.

Failure databases from real networks can be imported into Rice and "replayed" in the simulated environment either as a visualization of the EPN processes, or more complicated exploration of the failures context (e.g. simulation of several ways of a failure recovery and their effects).

The Rice system includes a rich library for support of fast system customization and implementation of a particular power system. Parts of this library implement basic power flow operations, standard agents behaviour, etc. All inter-agent communications are based on the KQML language.

In this section, we will shortly describe the basic principles of a power system components implementation in the Rice system. We will use the terminology common in the field of a multi-agent software development.

Almost every agent property is defined in the way how it changes the agent's interaction and communication ("behaviour") with its surroundings. A typical agent definition consists of three basic classes of its behaviour:

Event-driven actions are triggered in the case of an important environment change. For our topic these kind of events are important: energy flow (energy demand) change, facility error (resulting in outage or not), outage caused by other facility error. To satisfy the multi-agent design principles and make the development process clear, we must distinguish well before the two latter events.

Note that there is no mechanism to simulate the outage propagation as is. This is not needed and should not be implemented. Instead, we can see every agent as an autonomous entity which reacts to the event "my input lines are at zero level." The simplest reaction is to propagate the outage state to all agent's outputs, but more sophisticated agents that will search through their inputs and ask every peer if it can supply more power to overcome the outage can be implemented.

Using complicated behaviour implementation a very complex and even conventionally hardly computable error propagation throughout the power system can be modelled even if the behaviour of all engaged agents can be highly dynamic and state-dependent. For

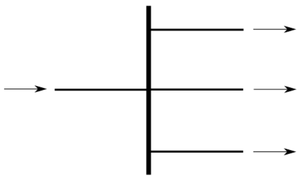


Figure 5.11: Simple bus

instance the topology of the network can vary in the dependence of the power lines load, daytime, etc.

Time-based actions are special types of the event-driven actions. They are triggered (1) by approaching an absolute time or (2) after expiration of a time period (relative time). In the real implementation the model must incorporate various time-based actions: "Service period reached," "Critical outage time reached" or "End of the fiscal period reached." All of them (plus some others) have built-in support in Rice.

Idle-loop actions (ILA) are actions which should be performed *continuously* (or *as frequently as possible*). Note that Rice does not have discrete time (time flow in exact steps) and so time between ILA triggers can vary. If the simulation needs exact time slices, the time-based actions with simulator time speed appropriate to the hardware performance are to be used.

5.3.1 Example Nodes Implementation

In this section, we will demonstrate the process of an agent behaviour definition on two examples. We will concentrate on the process philosophy and marginally on the practical implementation and particular code.

5.3.1.1 Simple Bus

In the first example, we will demonstrate the main guidelines of a simple bus agent definition (the Figure 5.11). The bus has one input and three outputs.

Power demands originating from outputs are sent to the input. In case that the demand exceeds the capacity of the source, the demanding agent must be informed that the demand can not be satisfied. The agent will not simulate over-voltage.

Outage on the input will be propagated to all outputs. There is no backup so no special action will be taken.

We will define two time constants as a *maintenance period* and *maintenance duration*. Before the maintenance the agent will send a message to its outputs that there is a maintenance outage scheduled. This makes it possible for the outputs to establish any other power source, intelligently change the network topology and overcome the outage.

According to this design plan we must define these event-driven actions:

1. power demand arrived from output: check if the saturated demand will exceed the overall demand limit. Send granting or denying message.
2. outage on input: send outage message to all of the outputs and set the agent to the "error" state.[6]

The first action will be defined by only a few lines of code.[7] Rice has a built-in KQML parser which will launch the agent function corresponding to the arrived message. Function *powerDemand* is launched when the power demand KQML message arrives. The code follows:

```
def powerDemand(self,msg,demand):
 if self.powerConsumed + demand > self.maxDemand:
   utils.kqml.send(msg.sender,
                   PyMessage("receiver.denied()"))
 else
   self.powerConsumed+=demand
   utils.kqml.send(msg.sender,
                   PyMessage("receiver.granted()"))
```

[6]There is a principle in the multi-agent programming saying that any agent should not set the state of another agent. Instead of this every agent decides on its own about its state. We silently conform to this notion here: sending the outage message to the outputs does not mean that the output agents will be set in outage state too. They can have some mechanisms to overcome the outage (power backup, intelligent source searching etc.) and so they decide how to react to this message on their own.

[7]in the Python programming language

All KQML message constitution, sending, delivering and parsing is managed in the Rice standard library. All we have to define is just the core logic.

The second action will be defined in this way:

```
def rLAgentStatus(self,status):
  if self.msg.inReplyTo=="inputStatus":
    print "Input status is:  '%s'" % status
    # no power backup source here – just copy the input status
    self.setStatus(status)
    # inform all outputs about status change
    for sub in self.outputSubscriptions:
        sub.reply("receiver.rLAgentStatus('%s')" % status)
```

rLAgentStatus is a function launched when the message about the agent status has arrived. By examining the *in-reply-to* KQML field we verify that the status change is related to the input (in this example it is not really necessary but it is in the cases where the agent has more inputs).

For informing the outputs we use standard KQML tool – *subscriptions*. The agent that wants an information about status changes from some other agents sends a *subscription demand for status* message to that agent. Subscription message means that the agent wants to be informed about all changes of some value in the other agent (in this case the status). So when a status change occurs, the agent informs all agents which subscribed for this kind of information. This mechanism realizes other principle of the multi-agent design: only the demanded (and so really needed) information are communicated between agents.

The last function in the example is the time-based maintenance. The implementation uses Rice built-in functions.

```
def maintenanceUpcomming(self,status):
 for sub in self.outputSubscriptions:
  sub.reply("receiver.rLAgentStatus('anticipating outage')")
 # wait for a defined time before the actual maintenance procedure
 utils.timer(self.maintWaitTime,self.maintenance)


def maintenance(self):
 for sub in self.outputSubscriptions:
  sub.reply("receiver.rLAgentStatus('outage')")
```

The function *maintenanceUpcomming* is launched in regular time periods. It informs all outputs about the scheduled maintenance and sets the

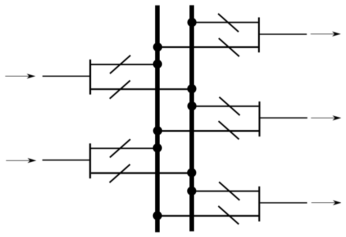


Figure 5.12: Duplicate busbar

timer which will launch the function *maintenance* after the chosen time during which the agents connected to the output can look for a backup power source.

5.3.1.2 Duplicate Busbar

In this example, we have a typical duplicate busbar with two inputs and three outputs, see the Figure 5.12. Besides all actions defined in the previous example, we will provide a new action: if there is an outage on one input line, all outputs are switched to the other line (maximal demand checking is similar to the first example).

The principle of the agent implementation is similar to the previous one. With a minor customization we can use all actions defined above. The difference is mainly in the need to distinguish the messages originating from the first and the second input. We will use the *reply-with* KQML field for it. The subscription message will be sent by this message:

```
utils.kqml.send(self.input1,
    PyMessage("receiver.rLAgentStatus()",
       msgType=utils.kqml.subscription,
       replyWith="input1Status"))
```

By this message the agent subscribes to the information about all status changes of the agent *input1*. Incoming responses to this subscription will be tagged with the string *input1Status* in the *in-reply-to* KQML field.

Reaction to the outage on one line – switching all outputs to the second one will than be defined by this code:

```
def rLAgentStatus(self,status):
     if self.msg.inReplyTo=="input1Status":
        self.allToLine2()
     if self.msg.inReplyTo=="input2Status":
        self.allToLine1()
```

Function *allToLineX* will check if the overall demand does not exceed the maximal allowed demand and eventually propagate the outage similarly to the first example.

5.4 Modelling of Economic Aspects of a Power System Failure

Each electrical power system (EPS) failure causes a chain reaction of effects, all with negative economic impacts. The effects can be categorized a) from the point of view of the supplier and b) from the point of view of the customer. Both of these groups of effects can be further divided into parts denoted as *direct* and *indirect costs* [BA96]. The overall amount of costs arising as a result of a power system failure can be calculated as sum of all these costs. However, specification of these costs is a long and complicated process, since all these four groups consists of many items such as the cost of technology repair, the cost of unsold energy, the cost of technology restart (on the customer's side) or the cost of lost profits.

The ability to provide a *prediction* of these costs for a particular power system failure is thus a very welcomed feature of any computer system. The Rice system is able to provide a flexible multi-agent dynamic network platform for complicated simulation of all power network processes including the setup and computation of possible costs of particular effects of an outage within the network.

An interesting application of Rice comes out from the association with classical SCADA databases. These databases are primarily used to determine particular facility failure rates, various conditions of the failures, defective facility series etc. Such information is processed by means of the classical data mining and statistical analysis. In spite of that data mining is relatively advanced field, it can be hard or impossible to harvest some types of information from the database, where the entries map only particular failures, outages, etc.

But SCADA can be used for “profiling” particular facility type and its behaviour in the network which can be then used in the development of an agent simulating this facility. The particular methodology of this process must be defined yet and it is our plan for the near future.

We can also use a backward process. In Rice, agents send information about their state to the application which visualizes processes taking place in the network. The same principle can be used to create a *simSCADA* – simulated SCADA. By comparing data in the real SCADA and simSCADA, we can numerically evaluate the simulation accuracy.

Moreover, because failures are simulated by message passing, it is relatively simple to track down the failure origin and join this original failure with its consequences (measure precisely the amount of undelivered energy). This could be hard with classical SCADA and we consider this a promising feature of our system. It is simple to measure the overall sum of the undelivered energy and use this measure for the optimization of the agent behaviour principles.

The Rice framework offers several important features as regards the modelling of economic aspects of failures in the power supply:

- *reactive agents* – agent behaviour is determined by its inner state and its perceptions (i.e. the state of the agent’s environment).
- *local interactions* – interaction between agents is strictly in the “peer to peer” manner. There is no global authority or global synchronization.
- *on-demand information* – every interaction is initiated by an agent that needs a piece of information for its operation. This has two practical results: (1) At the time of the implementation we do not need to know which information will be necessary for which agent and so we do not have to pre-design data flows in the system. (2) Data flow rate is on the least possible level.
- *massive parallelism* – every agent in Rice consists of one or more threads or even separated applications. This means that no agent can be blocked by misbehaved peer.
- *non-discrete time* – some multi-agent systems are designed as “step-based” applications. This means that agents must work with small time slice operations and the whole operation flow is constructed

by a kind of time sharing algorithm. This causes serious practical impacts on possible agents design, coding style and programmer's skill. As we have mentioned above, Rice is massively parallel and so the agent design does not require to pay any special attention to the time sharing.

- *not necessarily real-time* – Rice can be used as a real-time system but a typical usage will be the simulation of a power system operation in a long time period (months or years) within a short-time period (accelerated simulation).

In this section, we will explain usage of the Rice system on the example of modelling a hypothetical part of an EPS. The purpose of the example is to present the process of a model implementation in the Rice system. In the example we will need two kinds of actions: (1) computation of actual outage cost and (2) computation of actual power demand.

The model has eight facilities: one power source (A), three transformation/distribution points (B,C,D), four consumer points (E,F,G,H) and seven power lines interconnecting the points (L1–L7). Visualisation of the network in the Rice-Viewer is displayed in the Figure 5.13.

In addition to the network topology, we must define the following behaviour principles for each agent:

- *power demand function* – amount of power consumed by the facility in the time
- *reliability function* – frequency and duration of the facility errors in the time
- *reaction to inputs outages* – means of reactions can vary. Basic agents set their state to the input outage, set their output level to zero power or start to malfunction. Intelligent agents can start searching for new inputs configuration to re-achieve full energy supply from other sources.
- *loss function* – defines the amount of immediate and cumulative lost profits regarding the outages and other malfunctions of the facility. As well as with other functions, the lost profits depend on the day time, the day of the week, etc.

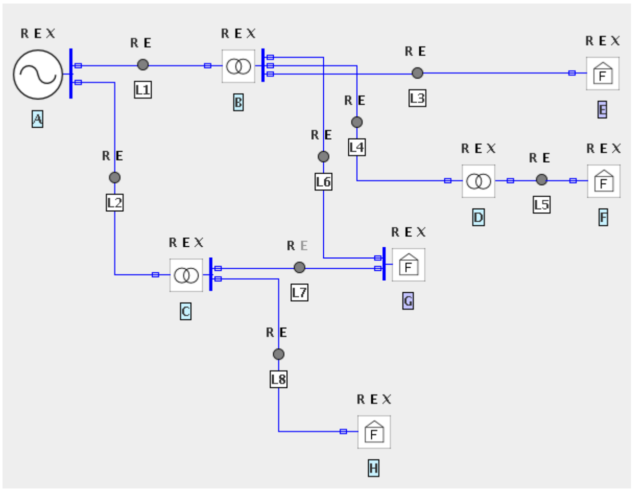


Figure 5.13: The example model of EPS used for computation of economic losses.

5.4.1 Network Parts

The parts of our simulated power network are equipped with the following parameters.

5.4.1.1 Source

In our example we consider a particular energy source **A** with constant available energy (1.5 MW). Demands exceeding this limit will be simply rejected (possibly causing outage in the demanding facility). The source itself is maximally reliable (possibility of an self-caused outage is zero).

5.4.1.2 Distribution Points

We have three distribution points (DP) of different kind. First DP **B** is considered to be duplicated and so inner-fault-free. Therefore its reliability is maximal. Second DP **C** is dedicated to supply power for two factories so its reliability must be at good level. Let it be one error a year, with average duration of 20 minutes, Gaussian distribution of the outage duration with standard deviation 5. But DP **C** has another limitation – the total output power can be only 0.7 MW. Demands exceeding this limit will lead to denying the demand.

Last DP **D** is an old local facility dedicated to city quarter **F**. The **D**'s error level is high (4 errors a year). Time to recover is long (average 3 hours) and very varying (deviation 1.5).

5.4.1.3 Power Lines

In the Rice system, there are separate agents for each power line. In our example we consider all power lines fault free. There is only one cycle in the network graph: A-B-G-C-A, so there are two ways of delivering power from **A** to **G**: A-B-G and A-C-G. The latter one is used as the network initial configuration.

5.4.1.4 Consumer Points

There are four consumer points (CP). CP **E** and **H** are factories with middle-cost production. Both factories use no machinery which can be destroyed in the case of an outage. The lost profits of the factories can be approximated with a linear function as loss of \$3 000 per hour (factory **E**) or \$1 000 per hour (factory **H**). The average power consumption of the factories is 100 kW for factory **E** and 200 kW for factory **H**. Power consumption changes in steps with the Gaussian distribution with the mean of 1 kW and the deviation of 0.1. Both factories do not run overnight, so we must consider power consumption deviation throughout the day and count a loss only when the factory is running (outages in the not-running time are considered zero). Factory **E** has old internal power lines which can cause rare internal outage (average rate is 0.1 per year, duration of 5 hours with the deviation of 0.01).

Factory **G** is larger than the previous ones and has outages-sensitive high-cost technology. Even short outages can cause high money losses because of damaged technology parts. That is why the loss function cannot be approximated linearly. Instead, we will use the following function for modelling of the loss level:

Outage length (t, in minutes)	**Loss function** (\$)
0–5	$30\,000 + (t * 1\,000)$
5–20	$35\,000 + (t * 700)$
> 20	$49\,000 + (t * 200)$

The factory is running overnight, all time with approximately the same power consumption (we can consider a consumption of non-machinery parts of the factory insignificant). The average value is 0.5 MW again with Gaussian changes with the mean of 3 kW and the deviation of 0.1.

The agent **G** is the only facility in the system which has two inputs. Therefore it is the only agent for which we must supply a special implementation of the reaction to input outages. The behaviour will be straightforward: to switch to another power supply line. In our first run of the simulation, the switch time (time to recover from the outage) will be 3 minutes and in the second run 10 minutes.

The last consumer point is the agent **F** which represents a city quarter. Hypothetical losses of this consumer are constant in different periods of time:

Daytime (hours)	**Loss function** (\$)
0am–8am	\$100 per hour
8am–5pm	\$300 per hour
5pm–12pm	\$500 per hour

The average power consumption of the agent **F** is 50 kW in the morning and 200 kW in the afternoon period of the day. The variation of the Gaussian changes is the constant of 3 kW.

5.4.2 The Agents Implementation

In our example, we use standard agent features whose templates are supplied by the Rice multi-agent framework libraries. In the agents implementation we can use standard functions and instantiate the agents classes. Rice supplies a function which runs each agents in its own thread. All functions in Rice are thread-safe.

Power lines are fault-free so we can simply instantiate standard class implementation:

```
    Line1=BaseLine(1, timer, HelperInterface.A_RL_LINE,
                   "L1", 0, "A", "B")
```

For other agents we will create corresponding classes which model particular agent behaviour. Rice uses event-driven model which means that a "hook" (trigger) function can be defined for each event processed by the agent. There will be only a few functions which are to be redefined. Some of them will be illustrated here (grouped by the behaviour class as stated above).

5.4.2.1 Event-Based Behaviour

Demand initialization – consumer agents (**E**, **F**, **G**, **H**) set their initial power demand value to the mean of their power consumption.

```
    class BaseConsumer(BaseFacility):
        def __init__(self, demandMean, demandVar,...)
        self.initPowerDemand = demandMean
        self.demandMean = demandMean
        self.demandVariation = demandVar
```

Reaction to outage – all agents (except **G**) will set itself into the *outage* state if the input lines are in the outage state, too. This is the standard behaviour. For the agent **G** a different action is implemented – try to switch its inputs:

```
  def outageBeginHook(self, cause):
    for input in inputs:
      res = input.agent.powerDemand(self.lastDemand)
      if not res==0:
        # power granted, status remains the same
        return
      # no power granted
      self.setRLStatus(StatusOutage(cause))
```

5.4.2.2 Idle-Loop Behaviour

We must continuously check the agent state and generate errors in particular time intervals. Also we want to simulate continuous changes of the

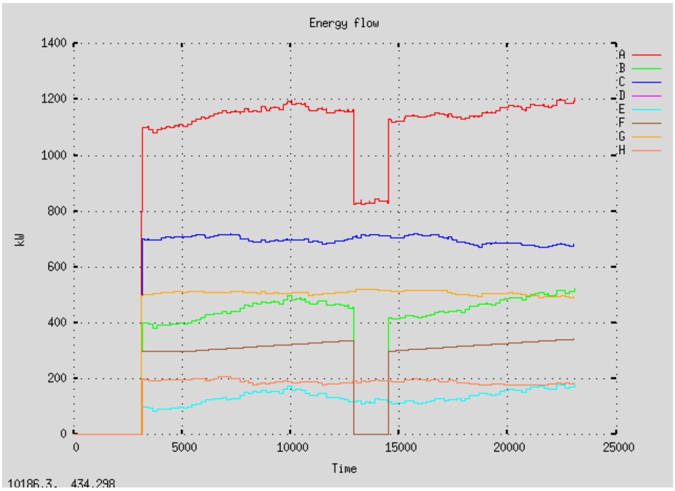


Figure 5.14: Power flow graph of part of the simulation of 10 years period produced by Rice.

power consumption. This will be performed by implementation of two hooks which are present in the standard Rice idle-loop implementation: *demandChangeHook* and *statusChangeHook*.

We will use standard Gaussian distribution of power consumption value changes. The corresponding function is supplied by the Rice Utils library.

```
def demandChangeHook(self, cause):
   return rice.Utils.gaussian(self.demandMean,
                               self.demandVariation)
```

To simulate the probability of facility errors we will use another standard function. Duration of the errors has also Gaussian distribution. The end of the error will be triggered by the timer.

```
def statusChangeHook(self, cause):
  if isinstance(self.rlStatus, StatusOK):
    res = rice.Utils.elapsed(self.timer,
                             self.errorProbab,
                              self.errorVar)
      if not res==0:
        self.setRLStatus(StatusError(cause=self.aid))
        self.setTimer(
          rice.Utils.gaussian(self.errorDurMean,
                              self.errorDurVar))
    # ...
```

These snippets of the example code illustrate that the implementation of a particular power system model within the Rice system is straightforward.[8]

5.4.3 The Simulation Run

The example simulation of the EPS used for computation of economic losses ran for ten years of the simulated time. The actual running time was 8 minutes. An example graph of power flow on the inputs of the agents in one period of the simulation can be seen in the Figure 5.14.

The resulting numbers and durations of power supply outages and the computed economic losses are summarized in the Table 5.5. Errors in the power supply from agent **D** cause only losses for agent **F**. **G** has two inputs, **B** and **C**. The initial input is **C**. When there is an error in the power delivery from **C**, the input is switched to **B** (which has a limited capacity). In the simulation, we have measured and computed the economic impacts of the two situations denoted as $\mathbf{G}_1$ (without using the **B** backup input) and $\mathbf{G}_2$ (with both inputs switching). Even if there are two inputs, the power supply would not be entirely uninterrupted because of the **B**'s supply limitation. The power restoration of **G** depends on the actual power consumption of agents **E** and **F**. We can see that the provision of another input for **G** has saved \$445 125. The time when **G** was switched to input **B** was too short to cause any overloading on **B** (denying of the demands of **E** and **F**). The agent **H** is not active for the whole day, which is why only 3 of its outages have caused an overall economic loss of \$1 183.

The nature of the simulation is strictly probabilistic. The resulting values vary in different runs of the simulation. In this one-run simulation the result was that the possible overall losses caused by not supplied

[8]The whole implementation of the example has about 200 lines of code in Python.

Table 5.5: The outages and economic loss during the simulation run

Agent	Outages time minutes	Economic loss
A	0	→B, →C
B	0	→E, →D, →G
C	213	→G, →H
D	7 970	→F
E	**262**	**$13 110**
F (total)	**7 970**	**$42 416**
F: 0am–8am	2 593	$4 321
F: 8am–17pm	2 015	$10 076
F: 17pm–24pm	3 362	$28 019
G_1 (one input)	**213**	**$555 321**
G_1: 0–5min out.	5	$34 510
G_1: 5–20min out.	34	$93 961
G_1: 20–∞min out.	174	$426 850
G_2 (two inputs)	**47**	**$110 196**
G_2: 0–5min out.	0	$0
G_2: 5–20min out.	20	$48 818
G_2: 20–∞min out.	27	$61 378
H	**213**	**$1 183**

energy will be $612 030 if the backup power line from **B** to **G** was not provided and only $166 905 if it was. We can see clearly how big the financial impact of an existence of such line is.

The number of intervening agents, events and actions in a real power system simulation is obviously higher than in the presented example with higher number of possible dynamic changes of the network topology. But we believe that the design of the Rice system allows us to cope with any such modelled situation – the next step lies in testing the system on a part of a real power network from the Ostrava region in the Czech Republic.

5.5 Future Work on the Rice System

We have described the design and prototype implementation of a multi-agent system for active simulation of energy flow in an electrical power network. The implementation framework meets all the required criteria which were identified in the process of power systems simulation analysis. It has already proven to be enough flexible, open, dynamic and robust even for possible real-time power systems monitoring.

The Rice system for simulation of power network processes provides a modular platform for analysis of the energy flows within any large-scale power system. During the design of the project, the system concentrates on handling of uncertainty either in the form of approximate values from the network or unknown behaviour of the take-off points.

The new assets that Rice brings to the field of power system simulation lie in the estimated calculation of the economic aspects of the power system outage, which is enabled by the modular design of the multi-agent network.

We have also described the first stages of implementing a natural language dialogue system for human-computer communication specialized to the domain of electrical power network field. The main purpose of the designed system is to support complicated queries of large time SCADA databases of electrical network failures included in a system for prediction and prevention of blackout and power network failures.

The text includes the description of building a corpus of electrical power networks texts as well as preliminary results of natural language processing analysis of the texts. These results show the need of adapting the current NL tools for the purpose of analysis of such specific domain. Extending the tools forms the baseline of our future directions on this topic of research.

The future directions of the implementation of the system lie mainly in testing the system with real-world data (which are actually very difficult to obtain due to their economic meaning). Nevertheless, we believe that the system can form an important acquisition with the ability to test and predict possible system outages of power networks.

Chapter 6

Conclusions and Future Directions

We have presented the current results of several research projects implemented in the NLP Centre at the Faculty of Informatics, Masaryk University. The central topics of all these projects are the syntactic and semantic analysis of natural language sentences with the concentration on the Czech language. The projects are in different stages of development, but all of them have already proved that they can significantly contribute to the field of computer processing of natural languages.

The Czech valency lexicon VerbaLex is still in its development phase, however, we can see its potential exploitation in large number of practical natural language processing applications, not only the syntactic parsing, but also problems like word sense disambiguation, predicate-argument structure extraction, machine translation or logical analysis. The linkage to the Princeton WordNet synsets provides the possibility of multilingual projections of VerbaLex to other languages. The experiments with using Czech valency frames for building Bulgarian and Romanian frames in Balkanet project have been evaluated as more than promising, see [TMBM06] and S. Koeva in [Chr04].

The usage of electronic dictionary resources in the computational linguistics has increased several times during the last decade due to evident reasons: 1) the computer equipment has reached such high state that it is feasible to process giga bytes of textual data [Gra03], and 2) the effectiveness gain is enormous when compared to the previous types of slow "manual" processing of linguistic information.

Despite this fact, the list of generally available dictionary writing systems (DWS) is not very wide – we may refer to e.g. Longman Dictionary Publishing System [McN03], TshwaneLex [JdS04] or the Dictionary Editor and Browser (DEB) [HPRR06, HP06a]. All the cited systems are based on XML databases that allow to capture practically any kind of structural data including monolingual and translational dictionaries, thesauri or encyclopediae. Longman DPS and TshwaneLex are self contained commercial applications that are designed for specific purposes – Longman DPS is used in the publishing house to bring out traditional paper based dictionaries as well as new electronic and on-line products. TshwaneLex on the other hand is a dictionary compilation system that allows to create and maintain several dictionary styles for various purposes. Both these systems are distributed on a commercial base.

The third system, the DEB platform, is an open source and freely available development framework developed at the Centre of Natural language processing at Masaryk University, Czech Republic. The system provides strict client server architecture for design of completely versatile dictionary applications. We have described the details of the system as a whole including the presentation of a new component, the Administration interface that is used by all client applications. The relevant features of the DEB platform are high modularity, configurability and flexibility which results in an easy adaptability for the various tasks. Thanks to them, the DEB platform represents a versatile base, on which the individual and powerful dictionary writing tools (clients) such as DEBVisDic are implemented.

In our view the XML formats within the DEB platform present a reasonable base not only for merging various lexical resources but also for their future standardization. This has been convincingly shown in the Balkanet Project where the VisDic XML format has been employed for building all 6 Balkanet languages (Bulgarian, Czech, Greek, Romanian, Serbian and Turkish) plus English. In fact, VisDic tool played the role of the instrument through which the first WordNet standardization steps have started. It is our belief that DEBVisDic tool can play the same role in the near future as well.

The DEB platform is being (and will be) thoroughly tested with its clients used in several teams all over the world. We may name e.g. the Dutch Cornetto project or the Polish and Hungarian WordNet projects. We also discuss the possibility of DEBVisDic being used as the main WordNet tool in the near future as well, namely in the preparation of the Global WordNet Grid.

The PRALED client is used in the Institute of Czech Language, Czech Academy of Sciences (Prague) as a dictionary writing system for building the Czech Lexical Database which is a large project planned for about 5 more years from now. The goal is to develop a lexical database of contemporary Czech containing approximately 100 000 entries.

Full parsing of natural language sentences is a complicated task that provides a transition from unstructured text into structural information suitable for all forms of information retrieval. Any kind of higher level language understanding and/or semantic processing must rely on the results of syntactic parsing. The quality of state-of-the-art syntactic parsers [Cha01, Col99] is still not completely satisfiable even for analytical languages like English. In case of free-word-order languages like Czech, the situation is even more complicated.

At the Centre of Natural Language Processing at the Faculty of Informatics, Masaryk University in Brno, a full syntactic parsing system is being developed since 1997. The parsing system `synt` currently provides fast and reliable algorithm that provides basis for higher-level structural processing of common natural language texts. Nowadays, the methods for best analysis selection are being implemented in the system – some of these methods were in detail presented in this text. We believe that by improving the `synt` tree ranking algorithm with these methods, we will have one of the best Czech parsers at hand for the needs of the Normal Translation Algorithm for automatic semantic analysis of Czech sentences by means of the Transparent Intensional Logic (TIL).

Pavel Tichý published his views on the language analysis in TIL in several papers (e.g. [Tic94a]) and started to write a thorough work on that matter [Tic94b]. However, he managed to write only the first out of the intended twelve chapters, and thus has left a lot of particular phenomena of natural language without the prescription of their proper analysis. The first text, that tries to describe the algorithm of translation of natural language sentence in its completeness, is the cited work [Hor02]. Nevertheless, the algorithm is still in its first version only and its finalization is the work of Tichý's followers such as Materna, Oddie, Duží, Horák and others.

The sentence analysis using TIL in our work is directed not only to the mere meaning representation, but also provides a systematic basic for automatic logical inference of new facts. Tichý has proposed a theoretical framework for the TIL inference in [Tic82], which is further discussed and elaborated in [DJM08]. In a long term NLP Centre project, we are developing particular methods for a question-answering mechanism based on knowledge base data structures from [Hor02, chapter 6]. The first prototype implementation of such system have already been published in [GH07a] and [GH07b].

In the last chapter, we have presented the Rice multi-agent system that allows working with the power network processes without the need of any of the network component but still with the power to define and capture any complex behaviour of the components in real situations including the possibility of automatic estimation of the economic loss caused by a network failure.

The described communication framework follows all the required criteria which were identified in the process of power systems simulation analysis. It has already proven to be flexible enough, open, dynamic and robust even for possible real-time power systems monitoring. This is continuously validated by the growing implementation and it will be tested in cooperation with large Czech power supply companies. The system architecture is suitable for usage in largely decentralized networks of autonomous data gathering and processing units. Especially an integration with networks of thin-client sensors (e.g. ZigBee [Zig05]) deserves further attention because of the promising industrial applications.

Currently, the system exploits the valuable databases of power system failures which have been constructed for several years in 9 regions of the Czech Republic and Slovakia. Further development aims at integrating the results in the stochastic simulation of failures of particular network elements which, as we hope, will be an important step towards the tools and systems helping to effectively control the complex behavior of such emergent systems.

Bibliography

[Aea93] E. Abel et al. A Multiagent Approach to Analyze Disturbances in Electrical Networks. In *Proceedings of the Fourth ESAP Conference*, pages 606–611, Melbourne, Australia, 1993. 160

[AH02] J. Aycock and R.N. Horspool. Practical Earley Parsing. *The Computer Journal*, 45(6):620–630, 2002. 86

[Ami01] M. Amin. Toward self-healing energy infrastructure systems. *the IEEE Computer Applications in Power journal*, 14, part 1:20–28, 2001. 162

[BA96] R. Billinton and R.N. Allan. *Reliability Evaluation of Power Systems.* Plenum Press, New York, second edition, 1996. 196

[Bal04] The Balkanet Project Website. `http://www.ceid.upatras.gr/Balkanet/`, 2004. 8, 10

[BBHS+04] S. Baumann, C. Brinckmann, S. Hansen-Schirra, et al. The MULI Project: Annotation and Analysis of Information Structure in German and English. In *Proceedings of the LREC 2004 Conference*, Lisboa, Portugal, 2004. 69

[BBR87] G. E. Barton, R. C. Berwick, and E. S. Ristad. *Computational complexity and natural language.* MIT Press, Cambridge, Massachusetts, 1987. 85

[BKPR06] M. Baroni, A. Killgarriff, J. Pomikálek, and P. Rychlý. Webbootcat: a web tool for instant corpora. In *Proceeding of the EuraLex Conference 2006*, pages 123–132, Italy, 2006. Edizioni dell'Orso s.r.l. 180

[BN00] H. Bunt and A. Nijholt, editors. *Advances in Probabilistic and Other Parsing Technologies.* Kluwer Academic Publishers, 2000. 70, 90

[Bod03] Rens Bod. An efficient implementation of a new DOP model. In *EACL 2003*, pages 19–26, 2003. 76

[BPGR06] Hans C. Boas, Elias Ponvert, Mario Guajardo, and Sumeet Rao. The current status of German FrameNet. In *SALSA workshop at the University of the Saarland*, Saarbrucken, Germany, 2006. 14

[BS05] Gregory Burd and Kimbro Staken. Use a Native XML Database for Your XML Data. *XML Journal*, May. 26, 2005, 2005. `http://xml.sys-con.com/read/90126.htm`. 40

[BSDH98] Srinivas Bangalore, Anoop Sarkar, Christine Doran, and Beth Ann Hockey. Grammar & Parser Evaluation in the XTAG Project, 1998. `http://www.cs.sfu.ca/~anoop/papers/pdf/eval-final.pdf`. 115

[BSS98] A. Barceinas, J. A. Sánchez, and J. L. Schnase. MICK: A KQML inter-agent communication framework in a digital library. In *Memorias del Simposium Internacional de Computación (CIC'98)*, Mexico City, 1998. 167

[C^+04] K. Czajkowski et al. The WS-Resource Framework, 2004. `http://www.globus.org/wsrf/specs/ws-wsrf.pdf`. 160

[CD02] Michael Collins and Nigel Duffy. New Ranking Algorithms for Parsing and Tagging: Kernels over Discrete Structures, and the Voted Perceptron. In *ACL 2002*, pages 263–270, 2002. 76

[CJa02] Object Management Group. *IDL to Java Language Mapping Specification.*, 2002. `http://www.omg.org/technology/documents/formal/omg_idl_to_java_language_mapping.htm`. 173

[CNC06] The Czech National Corpus, 2006. `http://ucnk.ff.cuni.cz/english/`. 7

[COB04] Institut de Recherche en Informatique de Toulouse. *The COBALT project*, 2004. `http://www.irit.fr/recherches/SIERA/GRS/coop.frame.shtml`. 167

[Col98] Michael Collins. dep2phr – conversion between dependency and phrase structures, 1998. `http://ufal.mff.cuni.cz/pdt/Utilities/dep2phr/`. 110, 112

[Col99] M. Collins. *Head-driven Statistical Models for Natural Language Parsing.* PhD thesis, Computer Science Department, University of Pennsylvenia, Philadelphia, 1999. 209

[COR04] Object Management Group. *Common Object Request Broker Architecture Core Specification.*, 2004. `http://www.omg.org/technology/documents/formal/corba_iiop.htm`. 173

[CPy02] Object Management Group. *Python Language Mapping Specification.*, 2002. `http://www.omg.org/technology/documents/formal/python.htm`. 173

[Cre75] M.J. Cresswell. Hyperintensional Logic. *Studia Logica*, (34):25–38, 1975. 123

[Cre85] M.J. Cresswell. *Structured meanings.* MIT Press, Cambridge, Mass, 1985. 123

[Cro06] Douglas Crockford. JSON, The Fat-Free Alternative to XML. In *Proceedings of XML 2006*, Boston, USA, 2006. `http://www.json.org/xml.html`. 40

[CRZ03] Akmal B. Chaudhri, Awais Rashid, and Roberto Zicari. *XML Data Management: native XML and XML-enabled database systems.* Addison-Wesley Professional, 2003. `http://books.google.com/books?id=7LNhdOeQulQC`. 40

[CVJ92] D. Cockburn, L. Z. Varga, and N. R. Jennings. Cooperating Intelligent Systems for Electricity Distribution. In *Proceedings of BCS Expert Systems 92 Conference (Application Track)*, Cambridge, UK, 1992. 167

[Č$^+$83] F. Čermák et al. *Slovník české frazeologie a idiomatiky I-IV (Dictionary of Czech Phraseology and Idioms, SČFI).* Academia, Praha, 1983. 57, 61

[DBX07] Oracle Berkeley DB XML, 2007. `http://www.oracle.com/database/berkeley-db/xml`. 40

[Deb03] R. Debusmann. *A Parser System for Extensible Dependency Grammar*, pages 103–106. Loria, Nancy France, 2003. 71

[DJM08] Marie Duží, Bjorn Jespersen, and Pavel Materna. *Transparent Intensional Logic. Foundations and Applications.* Springer Science, the Netherlands, 2008. to appear. 121, 210

[DKPR98] Hoa Trang Dang, Karin Kipper, Martha Palmer, and Joseph Rosenzweig. Investigating regular sense extensions based on intersective Levin classes. In *Proceedings of Coling-ACL98*, Montreal CA, August 11-17, 1998. `www.cis.upenn.edu/~mpalmer/`. 13

[DTH99] L. DiPippo, B. Thuraisingham, and E. Hodys. Towards a Real-Time Agent Architecture – A Whitepaper. Technical report, 1999. TR99-273. 167

[Ear70] J. Earley. An efficient context-free parsing algorithm. In *Communications of the ACM*, volume 13, pages 94–102, 1970. 83

[Erl04] Jens Erlandsen. iLEX – an ergonomic and powerful tool combining, effective and flexible editing with easy and fast search and retrieval. In *EURALEX 2004*, Lorient, France, 2004. demonstration. 9

[Eur99] The EuroWordNet Project Website. `http://www.illc.uva.nl/EuroWordNet/`, 1999. 8

[F$^+$95] J. Filipec et al. *Slovník spisovné češtiny (Dictionary of Literary Czech, SSČ)*. Academia, Praha, 1st edition, 1995. electronic version, LEDA, Praha. 56, 61

[FBS04] C.J. Fillmore, C.F. Baker, and H. Sato. FrameNet as a 'net'. In *Proceedings of Language Resources and Evaluation Conference (LREC 04)*, volume vol. 4, 1091-1094, Lisbon, 2004. ELRA. 14, 61

[FFMM94] T. Finin, R. Fritzson, D. McKay, and R. McEntire. KQML as an Agent Communication Language. In *Proc. of* 3$^{\text{rd}}$ *Int. Conf. On Information and Knowledge Management CIKM 1999*. ACM Press, 1994. 174

[Fis99] D. A. Fisher. Design and Implementation of EASEL. In *Proceedings of MacHack 14, the 14th Annual Conference for Leading Edge Developers*, Deeerborn, MI, 1999. 5, 153

[GAL03] Pablo Gamallo, Alexandre Agustini, and Gabriel P. Lopes. Learning Subcategorisation Information to Model a Grammar with Co-restrictions. *Traitement Automatique de la Langue*, 44(1):93–117, 2003. 91

[Gar07] A. Gardoň. The Object layer of the Dolphin database. `http://nlp.fi.muni.cz/projects/dolphin/object_layer.htm`, 2007. 146

[GF92] Michael R. Genesereth and Richard E. Fikes, editors. *Knowledge Interchange Format, Version 3.0, Reference Manual.* Stanford University, Stanford CA, USA, 1992. `http://www-ksl.stanford.edu/knowledge-sharing/kif/`. 176

[GH07a] Andrej Gardoň and Aleš Horák. Dolphin – a Knowledge Base for Transparent Intensional Logic. In *Proceedings of the Seventh International Workshop on Computational Semantics (IWCS-7)*, Tilburg, The Netherlands, 2007. 120, 210

[GH07b] Andrej Gardoň and Aleš Horák. The Learning and Question Answering Modes in the Dolphin System for the Transparent Intensional Logic. In *Proceedings of the First Workshop on Recent Ad-*

vances in Slavonic Natural Language Processing (RASLAN 2007), Brno, Czech Republic, 2007. 210

[Gra03] David Graff. English Gigaword. Technical Report LDC2003T05, Philadelphia, PA USA, 2003. 208

[GRA07] *GRASS – Geographic Resources Analysis Support System*, 2007. `http://grass.itc.it/`. 42

[GWA07] The Global WordNet Association. `http://www.globalwordnet.org/`, 2007. 9

[Haj04a] Jan Hajič. Complex Corpus Annotation: The Prague Dependency Treebank. Bratislava, Slovakia, 2004. Jazykovedný ústav Ľ. Štúra, SAV. 7, 87, 107, 110, 157

[Haj04b] Jan Hajič. *Disambiguation of Rich Inflection (Computational Morphology of Czech)*. Karolinum, Charles University Press, Prague, Czech Republic, 2004. 109

[Haj98] J. Hajič. Building a Syntactically Annotated Corpus: The Prague Dependency Treebank. In *Issues of Valency and Meaning*, pages 106–132, Prague, 1998. Karolinum. 75

[Han04] Patrick Hanks. Corpus Pattern Analysis. In *Proceedings of the Eleventh EURALEX International Congress*, Lorient, France, 2004. Universite de Bretagne-Sud. 14, 59

[Hav57] Bohumil Havránek, editor. *Příruční slovník jazyka českého (Reference Dictionary of Czech Language, PSJČ)*. Státní nakladatelství/SPN, Praha, 1933–1957. 56

[HCRT99] Jan Hajič, Michael Collins, Lance Ramshaw, and Christoph Tillmann. A Statistical Parser for Czech. In *Proceedings ACL'99*, Maryland, USA, 1999. 108

[HH05a] Dana Hlaváčková and Aleš Horák. VerbaLex – New Comprehensive Lexicon of Verb Valencies for Czech. In *Proceedings of the Computer Treatment of Slavic and East European Languages 2005*, pages 107–115, Bratislava, Slovakia, 2005. 10

[HH05b] Aleš Horák and Dana Hlaváčková. Transformation of WordNet Czech Valency Frames into Augmented VALLEX-1.0 Format. In *Proceedings of LTC 2005*, Poznan, Poland, 2005. 10

[HH07] J. Hajič and E. Hajičová. Some of Our Best Friends Are Statisticians. *Lecture Notes in Computer Science*, 4629:2–10, 2007. 3

[HHK06] Dana Hlaváčková, Aleš Horák, and Vladimír Kadlec. Exploitation of the VerbaLex Verb Valency Lexicon in the Syntactic Analysis of Czech. In *Proceedings of Text, Speech and Dialogue 2006*, pages 79–85, Brno, Czech Republic, 2006. Springer-Verlag. 71

[HHKK07] Aleš Horák, Tomáš Holan, Vladimír Kadlec, and Vojtěch Kovář. Dependency and Phrasal Parsers of the Czech Language: A Comparison. In *Proceedings of the 10th International Conference on Text, Speech and Dialogue*, Pilsen, Czech Republic, 2007. accepted for publication. 71, 106

[HK05a] Aleš Horák and Vladimír Kadlec. New Meta-grammar Constructs in Czech Language Parser synt. In *Proceedings of Text, Speech and Dialogue 2005*, pages 85–92, Karlovy Vary, Czech Republic, 2005. Springer-Verlag. 70, 71

[HK05b] Aleš Horák and Vladimír Kadlec. Czech Language Parsing using Meta-grammar Formalism with Contextual Constraints. In *Proceedings of the Computer Treatment of Slavic and East European Languages 2005*, pages 124–129, Bratislava, Slovakia, 2005. 71

[HK06] Aleš Horák and Vladimír Kadlec. Platform for Full-Syntax Grammar Development Using Meta-grammar Constructs. In *Proceedings of the 20th Pacific Asia Conference on Language, Information and Computation*, pages 311–318, Beijing, China, 2006. Tsinghua University Press. 71

[HKS02] A. Horák, V. Kadlec, and P. Smrž. Enhancing Best Analysis Selection and Parser Comparison. In *Lecture Notes in Artificial Intelligence, Proceedings of TSD 2002*, pages 461–467, Brno, Czech Republic, 2002. Springer Verlag. 71, 106

[HN05] K. Hall and V. Novák. Corrective modeling for non-projective dependency parsing. pages 42–51, 2005. 108, 111

[Hof95] B. Hoffman. *The Computational Analysis of the Syntax and Interpretation of Free Word Order in Turkish.* PhD thesis, University of Pennsylvania, Philadelphia, 1995. 69

[Hol04] Tomáš Holan. Tvorba závislostního syntaktického analyzátoru (Building a dependency syntactic analyser). In *Sborník semináře MIS 2004*. Matfyzpress, Prague, Czech Republic, 2004. In Czech. 108

[Hol05] Tomáš Holan. Genetické učení závislostních analyzátorů (Genetic learning of dependency analysers). In *Sborník semináře ITAT 2005*. UPJŠ, Košice, 2005. In Czech. 108

[Hor02] Aleš Horák. *The Normal Translation Algorithm in Transparent Intensional Logic for Czech.* PhD thesis, Faculty of Informatics, Masaryk University, Brno, 2002. iv, 4, 69, 119, 120, 121, 124, 134, 136, 140, 144, 150, 155, 209, 210

[Hor04] Aleš Horák. Types in Transparent Intensional Logic and Easel – a Comparison. In *Proceedings of the IASTED International Conference Artificial Intelligence and Applications 2004*, pages 833–837, Anaheim, Calgary, Zurich, 2004. The International Association of Science and Technology for Development. 120

[HP04] Aleš Horák and Karel Pala. Lexicons in TIL and Verb Valency Frames. In *Proceedings of the International Conference on Communications in Computing (CIC 2004)*, pages 255–261, Las Vegas, Nevada, USA, 2004. CSREA Press. 120

[HP06a] Aleš Horák and Karel Pala. DEB tools for merging linguistic resources. In *Proceedings of the Workshop on Layering Linguistic Information, LREC 2006*, pages 55–61, Italy, 2006. ELRA. 10, 208

[HP06b] A. Horák and M. Prýmek. Modelling of Economical Aspects of a Power System Failure. In *the Proceedings of the International Conference on Power, Energy, and Applications (PEA 2006)*, Gaborone, Botswana, 2006. 162

[HP06c] A. Horák and M. Prýmek. State-oriented Maintenance of Electrical Power Systems with Dynamic Multi-agent Network. In *Proceedings of CSIT 2006*, pages 310–316, Amman, Jordan, 2006. ASP University. 162

[HP07] Aleš Horák and Karel Pala. Building a Large Lexicon of Complex Valency Frames. In *Proceedings of the FRAME 2007: Building Frame Semantics Resources for Scandinavian and Baltic Languages*, pages 31–38, Tartu, Estonia, 2007. Lund University, Sweden. 10

[HPDM07] Aleš Horák, Karel Pala, Marie Duží, and Pavel Materna. Verb Valency Semantic Representation for Deep Linguistic Processing. In *Proceedings of the Workshop on Deep Linguistic Processing, ACL 2007*, pages 97–104, Prague, Czech Republic, 2007. the Association for Computational Linguistics. 120

[HPRP06] Aleš Horák, Karel Pala, Adam Rambousek, and Martin Povolný. First Version of New Client-Server WordNet Browsing and Editing Tool. In *Proceedings of the Third International WordNet Conference - GWC 2006*, pages 325–328, Jeju, South Korea, 2006. Masaryk University, Brno. 10

[HPRR06] Aleš Horák, Karel Pala, Adam Rambousek, and Pavel Rychlý. New clients for dictionary writing on the DEB platform. In *DWS 2006: Proceedings of the Fourth International Workshop on Dictionary Writings Systems*, pages 17–23, Italy, 2006. Lexical Computing Ltd., U.K. 10, 208

[HR07] Aleš Horák and Adam Rambousek. Dictionary Management System for the DEB Development Platform. In *Proceedings of the 4th International Workshop on Natural Language Processing and Cognitive Science (NLPCS, aka NLUCS)*, pages 129–138, Funchal, Portugal, 2007. INSTICC PRESS. 10

[HS00] A. Horák and P. Smrž. Large Scale Parsing of Czech. In *Proceedings of Efficiency in Large-Scale Parsing Systems Workshop, COLING'2000*, pages 43–50, Saarbrücken, 2000. Universität des Saarlandes. 77, 82, 85

[HS02] Aleš Horák and Pavel Smrž. Best Analysis Selection in Inflectional Languages. In *Proceedings of the 19th international conference on Computational linguistics*, pages 363–368, Taipei, Taiwan, 2002. Association for Computational Linguistics. 69, 90, 110

[HS03] Aleš Horák and Pavel Smrž. VisDic – WordNet Browsing and Editing Tool. In *Proceedings of the Second International WordNet Conference – GWC 2004*, pages 136–141, Brno, Czech Republic, 2003. 10, 25

[HS04] Aleš Horák and Pavel Smrž. New Features of WordNet Editor VisDic. In *Romanian Journal of Information Science and Technology*, volume 7(1–2), pages 1–13, 2004. 10, 39

[HVR08] Aleš Horák, Piek Vossen, and Adam Rambousek. A Distributed Database System for Developing Ontological and Lexical Resources in Harmony. In *Lecture Notes in Computer Science, Proceedings of CICLing 2008*, Haifa, Israel, 2008. Springer-Verlag. 10

[HŽ06] Tomáš Holan and Zdeněk Žabokrtský. Combining Czech Dependency Parsers. In *Lecture Notes in Artificial Intelligence, Proceedings of TSD 2006*, pages 95–102, Brno, Czech Republic, 2006. Springer Verlag. 108

[Cha01] E. Charniak. Immediate-head parsing for language models. In *Proceedings of the 39th Annual Meeting of the Association of Computational Linguistics*, Toulouse, France, 2001. 209

[Chr04] D. Christodoulakis. *Balkanet Final Report.* University of Patras, DBLAB, 2004. No. IST-2000-29388. 207

[ICU07] International Components for Unicode (ICU), 2007. `http://ibm.com/software/globalization/icu`, Open-source C/C++ and Java libraries for Unicode and I18N support. 42

[IDL02] Object Management Group. *OMG IDL Syntax and Semantics.*, 2002. `http://www.omg.org/cgi-bin/doc?formal/02-06-07`. 173

[JB03] N. R. Jennings and S. Bussmann. Agent-based control systems. Number 23(3), pages 61–74, 2003. 168

[JdS04] D. Joffe and G-M. de Schryver. TshwaneLex – Professional off-the-shelf lexicography software. In *Third International Workshop on Dictionary Writing Systems*, Brno, Czech Republic, 2004. Masaryk University, Faculty of Informatics. 9, 208

[Jen00] N. R. Jennings. On Agent-Based Software Engineering. *Artificial Intelligence*, 17(2):277–296, 2000. 162, 168

[JG02] T.F. Jaeger and V.A. Gerassimova. Bulgarian word order and the role of the direct object clitic in LFG. In M. Butt and T.H. King, editors, *Proceedings of the LFG02 Conference*, Stanford, 2002. CSLI Publications. 69

[Kay89] M. Kay. Algorithm Schemata and Data Structures in Syntactic Processing. In *Report CSL-80-12*, Palo Alto, California, 1989. Xerox PARC. 83

[KDP00] Karin Kipper, Hoa Trang Dang, and Martha Palmer. Class Based Construction of a Verb Lexicon. In *AAAI-2000 Seventeenth National Conference on Artificial Intelligence*, Austin TX, 2000. 14

[KH07] Vojtěch Kovář and Aleš Horák. Reducing the Number of Resulting Parsing Trees for the Czech Language Using the Beautified Chart Method. In *Proceedings of the 3rd Language & Technology Conference: Human Language Technologies as a Challenge for Computer Science and Linguistics*, Poznan, Poland, 2007. accepted for publication. 71

[KKH06] Vojtěch Kovář, Vladimír Kadlec, and Aleš Horák. Grammar Development for Czech Syntactic Parser with Corpus-based Techniques. In *Proceedings of Corpus Linguistic 2006*, pages 159–165, Saint-Petersburg, Russia, 2006. Saint-Petersburg State University. 71

[KKK00] J. Kocek, M. Kopřivová, and K. Kučera, editors. *Český národní korpus – úvod a příručka uživatele (The Czech National Corpus – introduction and user's handbook).* ÚČNK FF UK, Prague, 2000. 7

[KKRP06] Karin Kipper, Anna Korhonen, Neville Ryant, and Martha Palmer. Extensive Classifications of English verbs. In *Proceedings of the* 12th *EURALEX International Congress*, Turin, Italy, 2006. 134

[KMG01] A.K. Khan, R.P. Meters, and C.A. Los Gatos. Monitoring power for the future. *Power Engineering Journal*, 15(2):81–85, Apr 2001. 179

[KP+99] J. Kraus, V. Petráčková, et al. *Akademický slovník cizích slov (Academic Dictionary of Foreign Words, SCS).* Academia, Praha, 1999. electronic version, LEDA, Praha. 56, 61

[KRST04] Adam Kilgarriff, Pavel Rychlý, Pavel Smrž, and David Tugwell. The Sketch Engine. In *Proceedings of the Eleventh EURALEX International Congress*, pages 105–116, Lorient, France, 2004. Universite de Bretagne-Sud. 61, 135

[Kru03] G. Kruijff. 3-phase grammar learning. In *Proceedings of Workshop on Ideas and Strategies for Multilingual Grammar Development*, Vienna Austria, 2003. 71

[KS06] Vladimír Kadlec and Pavel Smrž. How Many Dots Are Really Needed for Head-Driven Chart Parsing? In *Proceedings of SOFSEM 2006*, pages 483–492, Czech Republic, 2006. Springer-Verlag. 83

[Lev93] Beth Levin, editor. *English Verb Classes and Alternations: A Preliminary Investigation.* The University of Chicago Press, Chicago, 1993. 13

[Lou98] M. Louw. Polaris User's Guide. Technical report, Lernout & Hauspie – Antwerp, Belgium, 1998. 8

[Mat04] P. Materna. *Conceptual Systems.* Logos Verlag, Berlin, 2004. 121

[Mat98] P. Materna. *Concepts and Objects*, volume 63 of *Acta Philosophica Fennica.* The Philosophical Society of Finland, Helsinki, 1998. 150

[MC07] Robert C. Moore and John Carroll. Parser Comparison – Context-Free Grammar (CFG) Data, 2007. `http://www.cogs.susx.ac.uk/lab/nlp/carroll/cfg-resources/`. 106

[McD06] Ryan McDonald. *Discriminative learning and spanning tree algorithms for dependency parsing.* PhD thesis, University of Pennsylvania, 2006. 108

[McN03] Michael McNamara. Dictionaries for all: XML to Final Product. In *XML Conference 2003*, Philadelphia, USA, 2003. 9, 208

[Mil90] G. Miller. Five Papers on WordNet. *International Journal of Lexicography*, 3(4), 1990. Special Issue. 8, 129

[MK91] J. T. Maxwell III and R. M. Kaplan. The Interface between Phrasal and Functional Constraints. In M. Rosner, C. J. Rupp, and R. Johnson, editors, *Proceedings of the Workshop on Constraint Propagation, Linguistic Description, and Computation*, pages 105–120. Instituto Dalle Molle IDSIA, Lugano, 1991. Also in Computational Linguistics, Vol. 19, No. 4, 571–590, 1994. 71

[MMdM99] I. Maks, W. Martin, and H. de Meerseman. *RBN Manual*, 1999. 61

[MMH03] D.J. McArthur, J.R. McDonald, and J. Hossack. *A Multi-Agent Approach to Power System Disturbance Diagnosis.* Springer-Verlag, Berlin, 2003. 167

[Mon74] R. Montague. *Formal Philosophy.* Yale University Press, New Haven, 1974. 4

[Moo00a] R. C. Moore. Improved Left-Corner Chart Parsing for Large Context-Free Grammars. In *Proceedings of the 6th IWPT*, pages 171–182, Trento, Italy, 2000. 70, 106

[Moo00b] R. C. Moore. Time as a Measure of Parsing Efficiency. In *Proceedings of Efficiency in Large-Scale Parsing Systems Workshop, COLING'2000*, pages 23–28, Saarbrucken: Universitaet des Saarlandes, 2000. 106

[Nev05] Z. Nevěřilová. Visual Browser. `http://nlp.fi.muni.cz/projects/visualbrowser/`, 2005. 56, 67

[NM04] M. Neteler and H. Mitasova. *Open Source GIS: A GRASS GIS Approach.* Kluwer Academic Pub, 2$^{\text{nd}}$ edition, 2004. 42

[NNH06] Jens Nilsson, Joakim Nivre, and Johan Hall. Graph Transformations in Data-Driven Dependency Parsing,. In *Proceedings of the 21st Conference on Computational Linguistics and 44th Annual Meeting of the ACL*, pages 257–264, Sydney, 2006. 108

[NP01] I. Niles and A. Pease. Towards a standard upper ontology. pages 2–9. ACM Press New York, NY, USA, 2001. 62

[NS04] T. Nagata and H. Sasaki. *A Multi-Agent Approach to Power System Restoration.* 2004. 167

[NT04] I. Niles and A. Terry. The MILO: A general-purpose, mid-level ontology. 2004. 62

[O$^+$02] Ian Oeschger et al. *Creating Applications with Mozilla.* O'Reilly and Associates, Inc., Sebastopol, California, 2002. 41

[OSI84] International Organization for Standardization. *Basic Reference Model for Open System Interconnection*, 1984. ISO 8072. 173

[P$^+$02] Jan Petr et al. *Slovník spisovného jazyka českého (Dictionary of Written Czech, SSJČ).* Academia, Praha, 1st edition, 2002. electronic version, created in the Institute of Czech Language, Czech Academy of Sciences Prague in cooperation with Faculty of Informatics, Masaryk University Brno. 56, 61

[PH05a] M. Prýmek and A. Horák. Design and Implementation of Multi-Agent System for Analysis of Electrical Power Networks. In *Proceedings of ElNet 2005 Workshop*, Ostrava, 2005. VSB Technical University of Ostrava. 162, 180

[PH05b] M. Prýmek and A. Horák. Multi-Agent Framework for Power Systems Simulation and Monitoring. In *Proceedings of ICICT 2005*, pages 525–538, Cairo, Egypt, 2005. Information Technology Institute, Giza. 162

[PH06a] Karel Pala and Aleš Horák. From WEB Pages to Dictionary: a Languge-independent Dictionary Writing System. In *Proceedings of the* 12th *EURALEX International Congress*, Turin, Italy, 2006. 10

[PH06b] Miroslav Prýmek and Aleš Horák. New Features in Power Networks Modelling Using the Rice System. In *Proceedings of ElNet 2006 Workshop*, Ostrava, 2006. VSB Technical University of Ostrava. 162

[PHL$^+$06] J. Pustejovsky, C. Havasi, J. Littman, A. Rumshisky, and M. Verhagen. Towards a Generative Lexical Resource: The Brandeis Semantic Ontology. In *Proceedings of LREC 2006*, Genoa, Italy, 2006. demo. 59

[PJ94] K. Pala and Všianský J. *Slovník českých synonym (Dictionary of Czech Synonyms, SČS).* Lidove Noviny Publishers, Praha, 1994. 56

[PP02] Tomáš Pavelek and Karel Pala. VisDic – A New Tool for WordNet Editing. In *Proceedings of the First International Global WordNet Conference*, Mysore, India, 2002. Central Institute of Indian Languages. 25

[PRS97] K. Pala, P. Rychlý, and P. Smrž. DESAM — annotated corpus for Czech. In *Proceedings of SOFSEM'97*, pages 523–530. Springer-

Verlag, 1997. Lecture Notes in Computer Science 1338. 7, 70, 86, 92

[Prý04] M. Prýmek. Vybrané problémy návrhu simulátorů umělého života a jejich řešení v systému Kairos (Selected Problems in Artificial Life Simulator Design and Their Solution in the Kairos System). In *Kognika a umělý život (Cognica and Artificial Life) IV*. Silesian University, Opava, Czech Republic, 2004. 164

[PSH01] D. O. Koval P. S. Hale, R. G. Arno. Analysis techniques for electrical and mechanical power systems. In *Proceedings of 2001 IEEE I&CPS Tech. Conf.*, pages 61–65, 2001. 179

[PŠ97] Karel Pala and Pavel Ševeček. Valence českých sloves (Valencies of Czech Verbs). In *Proceedings of Works of Philosophical Faculty at the University of Brno*, pages 41–54, Brno, 1997. Masaryk University. In Czech. 10

[Rad93] Andrew Radford. *Minimalist Syntax*. Cambridge University Press, Chicago, 1993. 69

[Ray07] T. Ray. The Tierra Simulator, 2007. `http://www.his.atr.jp/~ray/tierra/`. 164

[Reh03] C. Rehtanz. *Autonomous Systems and Intelligent Agents in Power System Control and Operation*. Springer-Verlag, Berlin, 2003. 160, 165

[RK07] Albena Rangelova and Jan Králík. Wider Framework of the Research Plan Creation of a Lexical Database of the Czech Language of the Beginning of the 21st Century. In *Proceedings of the Computer Treatment of Slavic and East European Languages 2007*, pages 209–217, Bratislava, Slovakia, 2007. 60

[RLN93] P.S. Rosenbloom, J.E. Laird, and A. Newell. *The Soar Papers: Readings on Integrated Intelligence*. MIT Press, Cambridge, MA, 1993. 142

[RM98] Louis Rosenfeld and Peter Morville. *Information Architecture for the World Wide Web*. O'Reilly and Associates, Inc., Sebastopol, California, 1998. 40

[RN03] S. Russel and P. Norvig. *Artificial Intelligence, A Modern Approach*. Pearson Education, Inc., Upper Saddle River, New Jersey, 2nd edition, 2003. 160

[RUB07] *The Ruby Programming Language*, 2007. `http://www.ruby-lang.org/en/`. 42

[Š04] Pavel Šmerk. Unsupervised Learning of Rules for Morphological Disambiguation. In *Lecture Notes in Artificial Intelligence 3206, Proceedings of Text, Speech and Dialogue 2004*, pages 211–216, Berlin, 2004. Springer-Verlag. 182

[Sam00] G. Sampson. A Proposal for Improving the Measurement of Parse Accuracy. *International Journal of Corpus Linguistics*, 5(01):53–68, 2000. 115

[Sam07] Geoffrey Sampson. Leaf-ancestor assessment software, 2007. `http://www.grsampson.net/Resources.html`. 115

[San04] Yohanes Santoso. Gnome's Guide to WEBrick, 2004. (`http://microjet.ath.cx/WebWiki/WEBrick.html`). 46

[SB03] G. Sampson and A. Babarczy. A test of the leaf-ancestor metric for parse accuracy. *Natural Language Engineering*, 9(04):365–380, 2003. 115

[SB95] SC Sciacca and WR Block. Advanced SCADA concepts. *Computer Applications in Power, IEEE*, 8(1):23–28, 1995. 161

[Sea69] J.R. Searle. *Speech Acts*. Cambridge University Press, Cambridge, 1969. 176

[Sed05] Radek Sedláček. *Morphemic Analyser for Czech*. PhD thesis, Masaryk University, Brno, Czech Republic, 2005. 71, 82, 109, 182

[SH98] P. Smrž and A. Horák. Determining Type of TIL Construction with Verb Valency Analyser. In *Proceedings of SOFSEM'98*, pages 429–436, Berlin, 1998. Springer-Verlag. 10

[SHKC05] Lukáš Svoboda, Aleš Horák, Vladimír Kadlec, and Pavel Cenek. Language Resources for Intelligent Processing of Dialogues about Electrical Networks. In *Proceedings of ElNet 2005 Workshop*, Ostrava, 2005. VSB Technical University of Ostrava. 162

[Sch07] Kim Schulz. *Hacking Vim: A Cookbook to get the Most out of the Latest Vim Editor*. PACKT Publishing, 2007. `www.vim.org`. 19

[Ski57] B. F. Skinner. *Verbal behaviour*. New York, 1957. 163

[SL00] Rohini Srihari and Wei Li. A Question Answering System Supported by Information Extraction. In *Proceedings of the 1st Meeting of the North American Chapter of the Association for Computational Linguistics*, pages 166–172, Seattle, U.S.A, 2000. 69

[SLŽ02] M. Straňáková-Lopatková and Z. Žabokrtský. Valency Dictionary of Czech Verbs: Complex Tectogrammatical Annotation. In C. Paz

Suárez Araujo M. González Rodríguez, editor, *LREC2002, Proceedings*, volume III, pages 949–956. ELRA, 2002. 11

[Smi90] B. Smith. *Towards a History of Speech Act Theory.* de Gruyter, Berlin/New York, 1990. 176

[SOA03] SOAP 1.2 Part 1: Messaging Framework, June 24 2003. `http://www.w3.org/TR/soap12-part1/`. 160

[SP04] Lukáš Svoboda and Lubomír Popelínský. Communication with WWW in Czech. *Cybernetica*, 40(3):349–363, 2004. 185

[SPHS05] P. Smrž, M. Prýmek, A. Horák, and A. Sinopalnikova. Emergent Systems and Intelligent Agents for Simulation of Power Systems,. In *Proceedings of ELNET 2004 Workshop*, Ostrava, 2005. VSB Technical University of Ostrava. 162

[Svo04] L. Svoboda. Multiword expression processing (in Czech). In Vojtěch Svátek, editor, *Poster Proceedings of the Znalosti 2004 Workshop*, 2004. 186

[TH01] D. Thomas and A. Hunt. *Programming Ruby.* Addison-Wesley Reading, MA, 2001. 42

[Tic04] P. Tichý. *Collected Papers in Logic and Philosophy.* Prague: Filosofia, Czech Academy of Sciences, and Dunedin: University of Otago Press, 2004. 4, 121

[Tic80] P. Tichý. The Semantics of Episodic Verbs. *Theoretical Linguistics*, 7:264–296, 1980. 136, 140

[Tic82] P. Tichý. Foundations of Partial Type Theory. *Reports on Mathematical Logic*, 14:52–72, 1982. 122, 155, 210

[Tic88] P. Tichý. *The Foundations of Frege's Logic.* de Gruyter, Berlin, New York, 1988. 4, 119, 120, 121

[Tic94a] P. Tichý. Cracking the Natural Language Code. *From the Logical Point of View*, III(2):6–19, 1994. 209

[Tic94b] P. Tichý. The Analysis of Natural Language. *From the Logical Point of View*, III, 2:42–80, 1994. 209

[TK98] J.C. Trueswell and A.E. Kim. How to prune a garden-path by nipping it in the bud: Fast-priming of verb argument structures. *Journal of Memory and Language*, (39):102–123, 1998. 90

[TMBM06] Dan Tufiş, Verginica Barbu Mititelu, Luigi Bozianu, and Cătălin Mihăilă. Romanian WordNet: New Developments and Applications. In *GWC 2006*, Jeju Island, Korea, 2006. Masaryk University, Brno. 207

[Tom86] M. Tomita. *Efficient Parsing for Natural Languages: A Fast Algorithm for Practical Systems.* Kluwer Academic Publishers, Boston, MA, 1986. 83

[Too07] Field Linguist's Toolbox, 2007. `http://www.sil.org/computing/toolbox/`. 9

[VB$^+$98] P. Vossen, L. Bloksma, et al. The EuroWordNet Base Concepts and Top Ontology. Technical Report Deliverable D017, D034, D036, WP5 EuroWordNet, LE2-4003, University of Amsterdam, 1998. 15, 129, 135

[vdBB02] A. van den Bosch and S. Buchholz. Shallow Parsing on the Basis of Words Only: A Case Study. In *ACL 2002*, pages 433–440, 2002. 76

[vdV03] Eric van der Vlist. *RELAX NG.* O'Reilly Media, 2003. 45

[Vos98] Piek Vossen, editor. *EuroWordNet: A Multilingual Database with Lexical Semantic Networks for European Languages.* Kluwer Academic Publishers, Dordrecht, 1998. 8, 61

[Vyk05] Radek Vykydal. Nástroje pro vývoj gramatik přirozeného jazyka (Tools for Developing Natural Language Grammars). Master's thesis, Faculty of Informatics, Masaryk University, Brno, Czech Republic, 2005. In Czech. 73

[Vyk07] Radek Vykydal. The Grammar Development Workbench project, 2007. `http://nlp.fi.muni.cz/projects/grammar_workbench/manual-en/`. 76

[WN07] WordNet – a lexical database for the English language. `http://wordnet.princeton.edu/`, 2007. 8, 135

[Zig05] The ZigBee Alliance, 2005. `http://www.zigbee.org/`. 210

[Žab05] Zdeněk Žabokrtský. *Valency Lexicon of Czech Verbs.* PhD thesis, Faculty of Mathematics and Physics, Charles University in Prague, 2005. 20, 134

[ŽL04] Zdeněk Žabokrtský and Markéta Lopatková. Valency Frames of Czech Verbs in VALLEX 1.0. In Adam Meyers, editor, *HLT-NAACL 2004 Workshop: Frontiers in Corpus Annotation*, pages 70–77, May 2–7, 2004 2004. 7, 14

Annotation

Computer Processing of Czech Syntax and Semantics

In this text, we present results achieved in several natural language processing research projects of the NLP Centre, Faculty of Informatics, Masaryk University. The central topics of all these projects are the syntactic and semantic analysis of natural language (NL) sentences with the concentration on the Czech language. The leading person in these projects is Aleš Horák.

After an introduction, the second chapter describes three years of development of VerbaLex, a large lexicon of Czech verb valencies in the form of complex valency frames. This part is then followed by detailed descriptions of developed tools for working with this as well as other language resources. The presented tools are VisDic, DEBVisDic, DEBDict, PRALED and others. These tools are used by project teams all over the world.

The next chapter presents the latest development of the syntactic analyser `synt` that has been under development in the NLP Centre for several years. Besides the comprehensive description of `synt` inside and formats used, we also provide a comparison with several other natural language parsers, in which we show that the `synt` qualities are at least comparable to the best current parsers.

The fourth chapter outlines the advances made in the Normal Translation Algorithm (NTA) for Transparent Intensional Logic (TIL). It describes the methods and techniques aimed at an automatic translation from a NL sentence to its meaning expressed as a construction in TIL.

The description is not complete, yet, but we have concentrated on selected phenomena where we offer sample solutions or even prototypical implementations.

The last chapter gives details of a project that concentrates on intelligent methods for increasing the reliability of electrical networks. One its part involves the development of a human-machine communication framework for dialogues about the specific knowledge domain of electrical power systems (EPS). The task of another project part is the development of a multi-agent system for representing the EPS processes. These allow simulating different configurations of an EPS setup with an automatic computation of the economic aspects of the system failures.

Anotace (Czech Annotation): **Počítačové zpracování české syntaxe a sémantiky**

Tato práce prezentuje výsledky dosažené při řešení vybraných výzkumných projektů zpracování přirozeného jazyka v Centru ZPJ na Fakultě informatiky Masarykovy univerzity. Hlavním tématem těchto projektů je syntaktická a sémantická analýza vět přirozeného jazyka se zaměřením na češtinu. Vedoucím uvedených projektů je Aleš Horák.

Po úvodu je ve druhé kapitole popsána dosavadní tříletá práce na velkém lexikonu českých slovesných valencí VerbaLex, kde valence jsou uloženy ve formě tzv. komplexních valenčních rámců. Po této části následuje detailní popis vyvinutých nástrojů pro práci s tímto i jinými jazykovými zdroji. Jedná se o nástroje VisDic, DEBVisDic, DEBDict, PRALED a další. Tyto nástroje byly (a stále jsou) používány v jazykových výzkumných projektech po celém světě.

Následující kapitola ukazuje poslední vývoj syntaktického analyzátoru `synt`, který je jedním z dlouhodobých projektů Centra ZPJ. Kromě detailního popisu vnitřních technik a formátů systému `synt` uvádíme také srovnání s několika dalšími syntaktickými analyzátory přirozeného jazyka, kde ukazujeme, že `synt` je minimálně srovnatelný s nejlepšími současnými analyzátory.

Ve čtvrté kapitole shrnujeme pokrok ve vývoji Algoritmu normální translace (NTA) pro transparentní intenzionální logiku (TIL). Popisujeme zde metody a techniky pro automatický překlad věty přirozeného jazyka na její význam ve formě konstrukce transparentní intenzionální logiky. Uvedený popis není ještě kompletní, ale zaměřili jsme se na vybrané problematické jevy, u kterých uvádíme příklady řešení nebo dokonce prototypové implementace.

Poslední kapitola tohoto textu poskytuje detaily projektu zaměřeného na inteligentní metody pro zvýšení spolehlivosti elektrických sítí. Tento projekt zahrnuje jako jednu ze svých částí vývoj komunikačního rozhraní člověk-stroj pro dialogy ze specifické znalostní domény elektrorozvodných systémů (ERS). V další popsané části je vyvíjen multiagentní systém pro reprezentaci procesů ERS s možností simulace různých situací ERS s automatickým výpočtem ekonomických dopadů výpadků systému.

Book orders should be addressed to:

Librix.eu
Gorkeho 41
CZ-602 00 Brno, Czech Republic
E-mail: sales@librix.eu

Computer Processing of
Czech Syntax and Semantics

Aleš Horák

Published by Librix.eu, Gorkeho 41, 602 00 Brno, Czech Republic
in 2008

First edition, 2008

ISBN 978-80-7399-375-7