UNIVERSITY OF MISKOLC

FACULTY OF MECHANICAL ENGINEERING AND INFORMATICS

DOMAIN- AND LANGUAGE-ADAPTIVE NATURAL LANGUAGE CONTROLLING FRAMEWORK

Ph.D. Dissertation

AUTHOR:
Péter Barabás
MSc in Information Engineering

„JÓZSEF HATVANY" DOCTORAL SCHOOL
OF INFORMATION SCIENCE, ENGINEERING AND TECHNOLOGY
Research Area
APPLIED COMPUTATIONAL SCIENCE
Research Group
DATA AND KNOWLEDGE BASES, KNOWLEDGE INTENSIVE SYSTEMS

Miskolc, 2013

DOI: 10.14750/ME.2013.009

Declaration

The author hereby declares that this thesis has not been submitted, either in the same or in different form, to this or to any other university for obtaining Ph.D. degree.

The author confirms that the submitted work is his own and the appropriate credit has been given where reference has been made to the work of others.

Nyilatkozat

Alulírott Barabás Péter kijelentem, hogy ezt a doktori értekezést magam készítettem, és abban csak a megadott forrásokat használtam fel. Minden olyan részt, amelyet szó szerinti vagy azonos tartalomban, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Miskolc, 2013. június 3.

Barabás Péter

A disszertáció bírálatai és a védésről készült jegyzőkönyv megtekinthető a Miskolci Egyetem Gépészmérnöki és Informatikai Karának Dékáni Hivatalában, valamint a doktori iskola weboldalán az Értekezések menüpont alatt: *http://www.hjphd.iit.uni-miskolc.hu*.

DOI: 10.14750/ME.2013.009

Témavezető ajánlása

Barabás Péter: "Domain- and Language-adaptive Natural Language Controlling Framework" című PhD értekezéséhez

Barabás Péter a szoftverfejlesztés és információ menedzsment területének nagy tapasztalattal rendelkező szakértője. A doktori dolgozatában olyan területet választott, ahol az elméleti eredmények a gyakorlat számára is nagy jelentőséggel bírnak és a kidolgozott javaslatok hatékonyan átültethetőek a gyakorlatba.

A kiválasztott feladat, a természetes nyelvi ember-gép interfészrendszerek fejlesztése napjaink egyik aktuális témaköre, melynek számos helyi vonatkozása van. A vizsgált rendszer kapcsolódik a Hatvany József Informatikai Tudományok Doktori Iskola korábbi hallgatóinak munkáihoz, ezen belül is elsősorban Dr. Baksáné Dr. Varga Erika tevékenységéhez. Barabás Péter dolgozatában a Varga Erika által kidolgozott alapmodellt sikerrel kiterjesztette és azt a gyakorlati alkalmazásokhoz szükséges optimalizált algoritmusokkal kibővítette. A megvalósított mintarendszerek teljesítményparaméterei azt mutatják, hogy a kidolgozott architektúra- és algoritmuskeret a valós méretű feladatokban is hatékony megoldást biztosít. A fontosabb alkalmazási területek közé tartozik a robotvezérlés, navigációs alkalmazások és az információ lekérdező rendszerek egyaránt.

Az értekezés tézisei és a témához kapcsolódóan megjelent publikációk igazolják, hogy a jelölt sikeresen végrehajtotta a kitűzött célt. A jelölt a kutatás eredményeiről rendszeresen beszámolt angol nyelvű folyóiratokban, illetve hazai és külföldi konferenciákon, ezáltal eleget téve a Hatvany József Informatikai Tudományok Doktori Iskola publikációs követelményeinek. Az eredmények igazolják, hogy a jelölt képes színvonalas, önálló kutatómunkára, munkáját a rendszeresség és a teljességre törekvés jellemzi. Maga az értekezés gondos és szerteágazó tudományos munkát, a szakirodalom alapos feltérképezését tükrözi. Az értekezés Barabás Péter saját eredményeit tartalmazza és a Hatvany József Informatikai Tudományok Doktori Iskola által megkövetelt tartalmi és formai követelményeknek mindenben megfelel. Mindezekre tekintettel a jelölt számára a Ph.D. cím odaítélését messzemenően támogatom.

Miskolc, 2013. június 3.

Dr. habil. Kovács László
tudományos vezető

DOI: 10.14750/ME.2013.009

Acknowledgements

The dissertation exposes many years of work which I could not be able to achieve without the support of others.

First of all, I owe my deepest gratitude to **Prof. Jenő Szigeti**, Head of "József Hatvany" Doctoral School for giving me support and the necessary permissions to the accomplishment of the doctoral procedure.

I am heartily thankful to my supervisor**, Dr. habil. László Kovács**, whose guidance and support from the initial to the final level enabled me to develop an understanding of the subject.

I am greatly thankful to my co-supervisor, **Prof. Dr. Imre Juhász**, whose professional knowledge and guidance helped me to implement my Ph.D. thesis.

I am indebted to many of my colleagues at the Department of Information Technology for helping me in organizing the events connected to the doctoral procedure.

At last but not least, this thesis would not have been possible without the unfailing encouragement and support of my family. I am especially grateful to my wife Ágnes, for her invaluable help in the completion of the project, and I truly appreciate the efforts of my parents who spared no pains to create an untroubled atmosphere during my working hours.

I dedicate this work to my son Péter, wishing him an unclouded childhood and power to make his dreams come true.

DOI: 10.14750/ME.2013.009

List of Abbreviations

AFD	Application Function Description
API	Application Programming Interface
ASR	Automatic Speech Recognition
CL	Concept Lattice
CRF	Conditional Random Field
DDL	Data Definition Language
DG	Dependency Grammar
DL	Descriptive Logic
DML	Data Manipulation Language
DTW	Dynamic Time Warping
FCA	Formal Concept Analysis
FDM	Function Description Model
FG	Formal Grammar
FOPL	First-Order Predicate Logic
FS	Function Signature
FSA	Finite-State Automate
FST	Finite State Transducer
GPS	Global Positioning System
GUI	Graphical User Interface
HCI	Human-Computer Interaction
HMI	Human-Machine Interface
HMM	Hidden Markov Model
HNLPG	Hungarian Natural Language Processing Group
HOPL	Higher-Order Predicate Logic
HTTP	HyperText Transfer Protocol
JSON	JavaScript Object Notation
KQML	Knowledge Query and Manipulation Language
LCRF	Linear-Conditional Random Field
LG	Link Grammar
NER	Named Entity Recognition
NL	Natural Language
NLC	Natural Language Controlling
NLI	Natural Language Interface
NLP	Natural Language Processing
NLTK	Natural Language ToolKit
NNS	Nearest Neighbor Search
OCR	Optical Character Recognition

DOI: 10.14750/ME.2013.009

OWL	Web Ontology Langauge
POI	Point Of Interests
POS	Part-Of-Speech
QLF	Quasi-Logical Form
RDF	Resource Description Framework
SDK	Software Development Kit
SDS	Speech Dialog System
SNLP	Stanford NLP
SNLPG	Stanford Natutal Language Processing Group
SPO	Subject-Predicate-Object
SRM	Semantic Representation Model
TAG	Tree Adjoining Grammar
TCP	Transmission Control Protocol
UIMA	Unstructured Information Management Applications
VP	Vantage Point
WG	Word Grammar
XML	eXtensible Markup Language
XSD	Xml Schema Definition

DOI: 10.14750/ME.2013.009

Contents

DOI: 10.14750/ME.2013.009

DOI: 10.14750/ME.2013.009

DOI: 10.14750/ME.2013.009

List of figures

DOI: 10.14750/ME.2013.009

List of tables

DOI: 10.14750/ME.2013.009

List of algorithms

DOI: 10.14750/ME.2013.009

1 Introduction

Communication with most of computer systems happens via user-friendly but not natural ways. People need fill out forms, click buttons, type instructions through a touch screen instead of saying commands in natural language to the system. Natural language processing is a more and more popular area of artificial intelligence such as robot controlling, automotive systems, navigation systems, etc.

The main interaction form between human and computer is speech in spoken dialog systems (SDS). Three types of such systems can be distinguished: state-based (Aust and Oerder 1995), (McTear 1997) and (McTear 1998) frame-based (Hulstijn et al. 1996), (Veldhuijzen van Zanten 1996) and agent-based (McTier 2002).

The state-based systems are the simplest and most commonly used. This kind of dialog systems represents series of states. In each state, the system asks for specific information from user. After every state is "filled in" the system can generate answer with several techniques e.g. calling functions or running external applications. In each state the processing of input is intent to particular and well-defined words. State-based approaches are used for simple tasks.

In more complex tasks frame-based techniques are used instead of states, like in (Hulstijn et al. 1996), (Veldhuijzen van Zanten 1996). A frame represents a task which has slots. A slot contains a piece of information that the system needs in order to complete the task. More slots can be filled in one time and the system can construct questions for empty slots. A slot can be marked as required or optional. If all required slots are filled the system can complete the task and can generate the answer.

McTier (McTier 2002) defines the agent-based system where a frame is filled cooperating with the user and the problem is attempted to be solved together with user. The system and user exchange knowledge and reason about their own actions and beliefs to fulfill previous tasks.

My goal is to define and implement a natural language controlling framework using frame-based dialog system which can be applied for robot controlling also. My framework should process text input which can be generated even by a speech-to-text converter. The primary supported language in our system is the Hungarian, which is very difficult to process because of its agglutinate property. The optimization of algorithms is also very essential to produce an accurate and quick responding system.

DOI: 10.14750/ME.2013.009

1.1 Preliminaries

1.1.1 Human-machine interfaces

The history of computer science is strongly characterized by the permanent development of user interaction methods. In the early beginning the dedicated monolith computer architecture was dominating where the client terminals could provide only simple character-based command interfaces. The only way for communication was the character terminal using special command languages. With the development of graphical terminals, the next phase was the graphic oriented command interface where a much easier understanding was provided for the users. The application of graphical user interface (GUI) elements improved significantly the efficiency of human-computer interaction (HCI). Nowadays, new interface channels are offered by the computer systems, like speech, motion or emotions. The dramatic change affects not only the dialog module, but the computer architecture itself has been changed significantly. The back-end part is distributed and heterogeneous. The servers are distributed in the cloud and the computers are embedded into different equipments, like robots, cars and household equipments. The development of intelligent machines is based on a huge software development background technology. A key component of this development tool is the human-machine interface (HMI) module.

A specific area of intelligent HMI is the ubiquitous computing. The term “ubiquitous computing" (Schmidt 2002) describes the phenomenon of interacting in context with artifacts and environments which are interwoven with processing and communication capabilities. The main characteristic of this architecture is the physical integration of the computers into the objects of the living and working environment and context. This architecture raises many methodological questions (Weiser 1993) and (Schmidt 2002). There are many related research directions targeting some special aspects, terms of ubiquitous computing like calm computing (Weiser and Brown 1998) and (Schmidt 2002), invisible computing (Norman 1999) and (Schmidt 2002), disappearing computer (Wejchert 2000), (Schmidt 2002) and context-aware computing (Schmidt 2002). The key modules in context-aware computations are context acquisition, context representation, context abstraction and adaption to context.

In (Cannan and Hu 2011), five main intelligent interaction categories are defined:

- speech (acoustic)
- optics
- bionics
- motion
- tactile.

The main task in acoustic category is speech recognition, where the spoken words are converted into text. There is a widespread potential application area of speech recognition for intelligent information systems. Beside speech processing, there are many other related

DOI: 10.14750/ME.2013.009

application areas like the area of acoustic myography, which is basically measuring the acoustic properties of muscles as they contract (Cannan and Hu 2011).

Another current trend in HMI is the development of multimodal interface. The multimodal interface means the usage of different collaborating communication channels in order to improve the functionality of the human-machine dialog. The architecture of the multimodal interface is presented, among others, in (Wei and Hu 2011), and the main structure is shown in Figure 1.1. Hands-free control could be considered as an important interface for disabled people. Hands-free HMI focuses on development of novel communication channels between a machine and the different parts of the human body (Wei and Hu 2011).

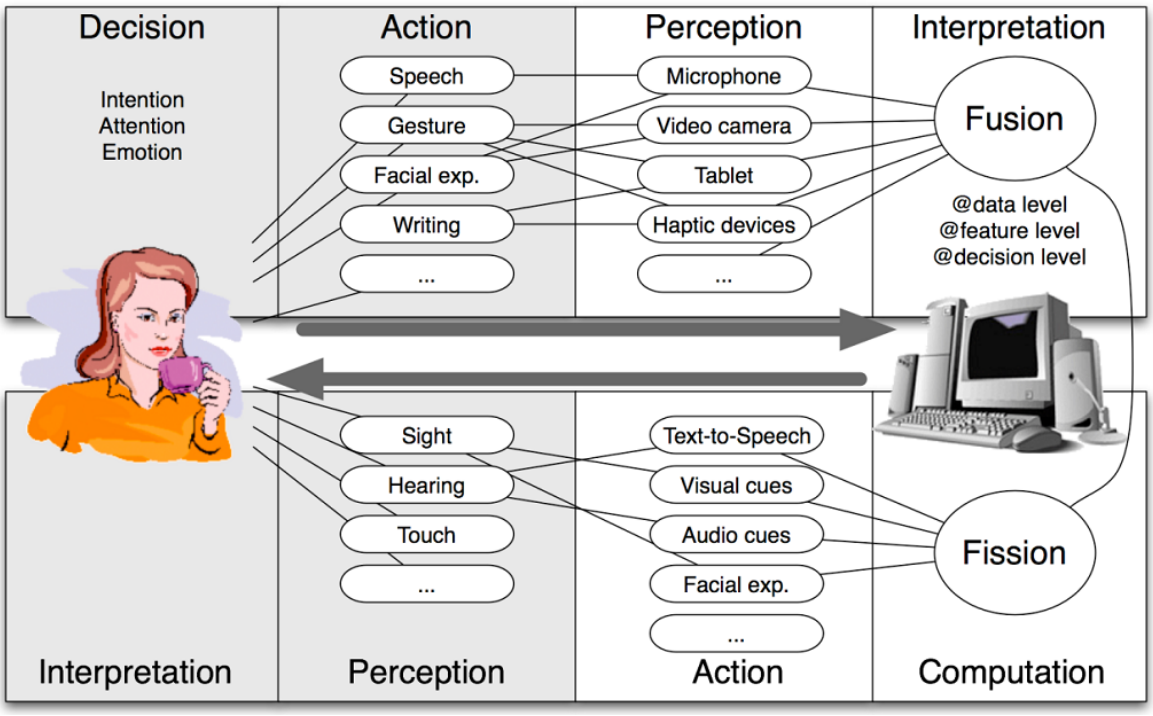

Figure 1.1. *Main structure of multimodel interface (source: (Wei and Hu 2011))*

Within the context of multimodal HMI, many new problems arise due to the integration issues. For example, the synchronization of speech and gesture is analyzed in (Ng-Thow-Hing and Pengcheng Luo 2010) in more details. The implemented framework initially configured by assignment of specific grammars to the different gesture types. After tagging the input text, the engine determines the best matching grammar patterns and selects the corresponding gesture model.

The main characteristics and current development trend for intelligent HMI frameworks are analyzed in (Puerta 1997). The key problems are related to following issues:

- applying novel methods, innovations and real life testing of hypotheses
- application of collaborative techniques
- development of human-oriented applications

DOI: 10.14750/ME.2013.009

- efficient interface development (reusability, modularity, standardization).

1.1.2 Semantic representations

The interpretation of an input text requires the capture of the semantic, the understanding the meaning of the sentences. In order to manage and operate on the semantic, some semantic representation model should be used. In the literature (Sowa 2000) there are four main knowledge representation methods:

- SPO models and semantic network models
- semantic frame models
- logic models
- rule based models.

Another differentiator issue is the characteristic of the quantitative data: there are deterministic and stochastic models (D'Argenio, Katoen, and Brinksma 1998).

In the text processing applications, the semantic network and the rule based models are the most widely used frameworks. Nowadays, the logic based systems gain more and more acceptance as it can provide a sound, theoretically proven background for the operations.

Regarding the logic models, the most important tools are the predicate logic and the descriptive logic. Predicate logic (Kowalski 1974) language consists of the following elements:

- data types
- functions with given signature
- terms
- predicates
- well-formed formulas (logic operators, quantifiers)
- variables.

In the first order predicate logic formalism (FOPL), the arguments of predicates are variables and terms. In the case of higher order predicate logic (HOPL), another predicates can be used as arguments too. For example, the sentence "Gabi knows that Zoli is reading a book" can be represented with the following higher order formula:

Know (Gabi, Read(Zoli, book)).

The predicate logic mechanism is a very general formalism, but the representation of temporal, dynamic and structural elements and constraints requires complicated formulas. For representation of object-class oriented semantic models, the descriptive logic (DL (Baader et al. 2003) is a suitable language. The DL consists of new constraint types to specify, among others, new relationship restrictions. The elements of DL are

$A \cap B$: intersection of two concepts

$A \cup B$: union of two concepts

DOI: 10.14750/ME.2013.009

$\neg A$: negation of a concept

$A \subseteq B$: subset, specialized concept

$\exists R.C$: existence on relationship

$\forall R.C$: all constrainsts on relationship

$\leq nR$: cardinality constraint on relationship

$\geq nR$: cardinality constraint on relationship.

The main shortcoming of DL is the absence of dynamic-, temporal- and object-level elements.

The frame-based model (Minsky 1975), (Varga 2011) unifies the structural and dynamic behavior components of the problem domain. This model has a very strong relationship with the object oriented approaches, were the class members correspond to slots of the frame and the methods are represented by behavior elements. A model contains a set of interconnected frames where the connection is implemented via relationship slots. The advantage of the frame model is the flexibility and the support of dynamic components. Its disadvantages are the ambiguity in representation and the absence of general constraint elements.

Semantic network is the base representation form were for many knowledge engineering applications. It was introduced in the late 1960's (Quillian 1968), (Varga 2011). The network contains of concept nodes and the edges correspond to different relationships among the concepts. The big advantages of this representation form are the great flexibility and the readability. This formalism enables the representation of any arbitrary binary relationship, like abstract relationships (specialization, containment,…) and any business level relationship (owner, supplier,...). The network supports the representation of both abstract concepts and instance level concepts. One of the main disadvantages of the model is the strict differences between node concepts and relationship concepts. The different variants of semantic networks are discussed, among others, in (Sowa 1992) and (Varga 2011).

On the field of text mining, the family of assertional semantic networks has a dominant role. One of main representatives of this group is the RDF graph (Klyne and Carroll 2004), (Varga 2011). The RDF graph is built up from triplets, where a triplet contains a subject, a predicate and an object component. It corresponds to an atomic information snippet. The graph corresponds to the set of related triplets. The RDF model enables the representation of higher level predicates too, as it contains intermediate resource node type and abstract node type. The representation of n-ary relationships requires a more complex formalism. The RDF does not support the definition of different constraints and the distinguishing of class and instance. An extension of RDF with instance level elements is the Conceptual Graph model, which enables the usage of a concept type hierarchy in the sense of (Kovács and Sieber 2009). This kind of semantic graph is widely used in computational linguistics where the verb is a central concept in the graph describing the meaning of a sentence. The concept of verb is connected with different relationships or roles to the other part of speech components (Sowa 2000), (Varga 2011).

DOI: 10.14750/ME.2013.009

One of the most powerful knowledge representation models is the OWL ontology model (Bechhofer et al. 2004). In ontology the structural part is based on a concept graph providing different elements for abstract concepts and instance concepts. The model contains a module for constraints given in descriptive logic language. Thus, an ontology engine can provide the validation of the models as well as the reasoning from different logic rules.

1.2 Aims and scope

The main goal of our research is to develop a natural language controlling framework which uses mostly rule-based approaches. The aim is to make such NLC system which can easily be adapted for several domains and for different languages as well. The implementation of framework should consider the alloying of existing solutions, working NLP engine modules into the framework components. The capabilities of the framework are fixed in advance which are:

1. **Domain-adaptivity:** the ability to easily learn concepts and relations of different domains without modifying inner the structure and workflows of framework
2. **Language-adaptivity:** the ability to parse natural language sentences in different languages with only teaching language-dependent parts of framework
3. **Extendibility:** the ability to extend the set of functions which wanted to be called by natural language commands
4. **Open interface:** the ability to reuse existing components of NLP engines and to implement, refine any part of framework for own needs.

In order to be able to achieve these objectives, the framework needs a semantic representation model (SRM) and a function description model (FDM) to satisfy the following requirements:

- main building blocks of the semantic model should be concepts and their relationships
- central concept should be the predicate
- apriori domain knowledge should be able to be represented with SRM to be capable of making sentence analysis
- SRM and FDM should be extended with in-sentence role data for proper function mapping
- it should provide high levels of flexibility and extendibility.

There are numerous NLP engines which provide solutions for basic text processing tasks like text parsing, spell checking, named entity recognitions, morphology analysis, sentence analysis, etc. These engines are regarded as utility libraries where I get ready-made algorithms, methods. The task of the developer is to integrate these solutions into the proposed framework with implementing provided interfaces.

The tasks of the project can be summarized as follows

DOI: 10.14750/ME.2013.009

1. The first task of the project is to create the structure of natural language controlling framework considering domain- and language-adaptivity and alloying exiting NLP engines.
2. The second task is to develop semantic models for representing domain knowledge for sentence analysis, application function descriptions for function mapping and algorithms to perform sentence analysis and function-mapping.
3. The third task is to find optimization problems in framework modules and to suggest solutions to achieve execution with lower costs.
4. Finally, the framework and two of its applications will be implemented for the verification of theoretical results.

1.3 Dissertation guide

Considering the tasks to be solved the dissertation consists of the following chapters.

Chapter 2 introduces the topic of dissertation and summarizes the key concepts of NLP systems. At first, the levels and approaches of natural language processing was studied which is important to be able to determine which levels should be interpreted in the proposed system. The second stage is to overview of types of NLP application and the third is to learn about NLP tasks which should be studied how and which of them should be included in the developing framework. In last stage existing NLP frameworks were examined in many aspects: implemented tasks, NLC process and language support, application programming interfaces. The summary of the chapter contains an overview about the lack of the existing frameworks and the requirements of a novel approach for NLC frameworks.

Chapter 3 exposes the development of the natural language controlling framework highlighting the novel elements which fulfill the declared requirements. The chapter declares the modules of framework considering the language-dependency and the proper way of connecting them. The integration of existing "classical" NLP solutions are also examined and the correct way to use is proposed. In order to be able to separate language-dependent and language-independent modules a novel notation system is constructed which should contain words or suffixes in a universal form. With this supplement, different languages can be adapted to the system with implementing only the language dependent parts of framework. This chapter results in a new NLC framework structure with extension of a notation system.

Chapter 4 models the representation of domain knowledge and the application functions. The framework is planned to be used in any domain context with proper teaching of dependent parts. The incoming sentences need to be analyzed which is only possible if we have precognitions about the concepts of the domain. Therefore the adaptation means constructing the concepts of the domain and the relations between them. This chapter gives a solution as a domain knowledge representation form. The output of the NLC system is the execution of some real functions. To be able to extend applications with natural language control, the controllable functions need to be collected. The framework provides a well-defined

DOI: 10.14750/ME.2013.009

description form to define application functions. The last part of the chapter is to determine an algorithm which maps the sentence analysis result to the belonging application function description.

Chapter 5 studies optimization issues in framework components especially in sentence analysis and function mapping. Domain knowledge and function descriptions are built up in tree structures, so it is very important to be able to evaluate the cost of algorithms that manipulates them considering the increasing nodes of structures. The necessity of optimization is also explained by the real-time working mode of NLC-extended systems. The optimization concerns the knowledge and function description models, the algorithms and also gives recommendations to use specific types, structures, algorithms in implementation.

Chapter 6 introduces two sample applications: robot controlling and navigation application where the new NLC framework was taught and used. Teaching means constructing domain knowledge and function descriptions that are discussed in detail by concrete examples.

Chapter 7 summarizes new scientific results achieved in the project. It also outlines the directions of future research.

DOI: 10.14750/ME.2013.009

2 Background

The main research direction related current technologies on user interface focuses on the development of natural language interface (NLI). There are innumerous potential application areas of NLI, like search engines (semantic search), question answering (information system), controlling of robots or engines, text mining or machine translation systems. In the recent decades several approaches were developed to perform the natural language processing (NLP) efficiently. The main characteristics of syntax oriented (shallow) approaches (MacCartney 2009) that only lexical and syntactical similarities are measured during text processing. In this case the words of the text are assigned to dictionary entries with largest similarity like in the BOF model of Glickman (MacCartney 2009). The similarity is usually measured with statistical methods. Most of these methods use document texts without any annotation (Varga 2011). In some cases, the texts are annotated with grammatical information. The main problems of these simple approaches are that the unsupervised, not annotated methods do not work efficiently, and on the other hand, the grammatical annotation requires a large amount of investment.

The other direction uses a semantic oriented (deep) analysis. In this case, the sentences are transformed into some formal language expressions. The formal language framework can be used to perform reasoning and consistency validation (MacCartney 2009). The logic based approach can be used to manage many language elements like quantifiers but it cannot be used to describe some other language components like vagueness, idioms or aspect. The other main drawback of this approach is the efficiency: it can't manage open domain problems as the natural language is too complex for formal description.

The task of the NLP Engine is to accept user's command formulated in natural language sentences and convert these commands into low level program function calls. One of the first proposals on high level NLI engines is given in (Lee et al. 1998). The engine consists of the following components: NLP Module, Discourse Manager and KQML Call Generator as can be seen in Figure 2.1.

DOI: 10.14750/ME.2013.009

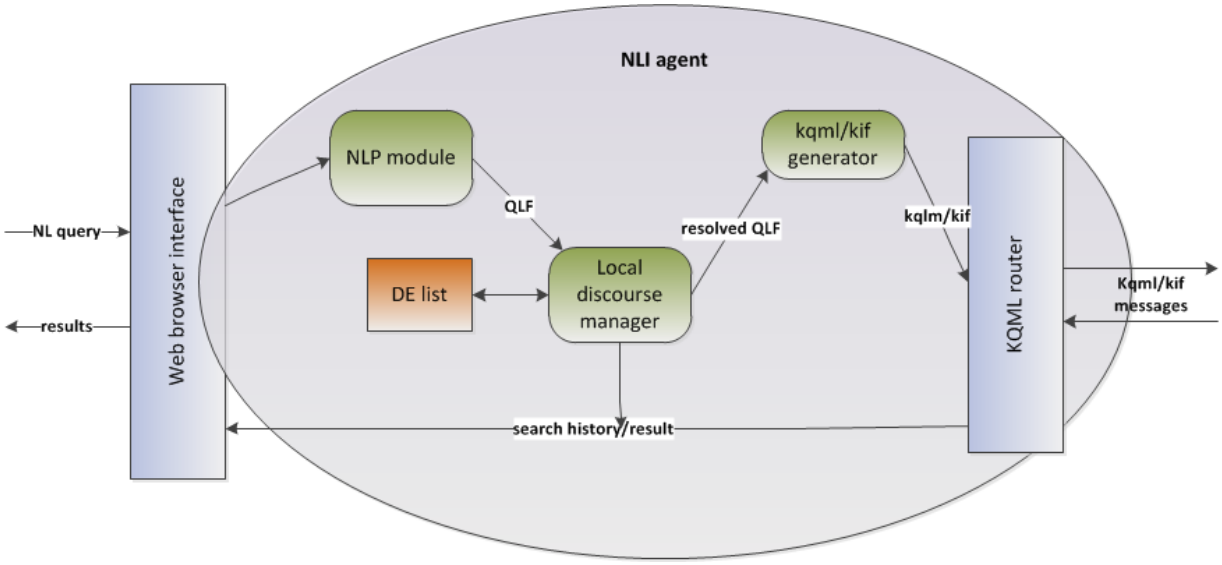

Figure 2.1. *The NLI Engine Structure* (Lee et al. 1998)

The NLP module converts the incoming sentences into QLF logical expressions. The first step of the processing is a morphological analysis. The engine determines the possible morpheme segmentations resulting word-level morpheme graphs. A dynamic programming approach is used to generate the morpheme graph. Based on a lexicon, the syntactic categories are determined using an extended categorical unification grammar formalism. The next step is a semantic analysis resulting in QLF expressions. The QLF language is based on extended first-order logic and it has predicate and argument list structure and objects are symbolized with terms. The QLF language (Alshawi and van Eijck 1989) contains temporal operators and second order arguments beside the usual quantifiers. The following example shows a sample sentence in QLF for the sentence "every representative voted":

quant(forall, x, Repr(x), past(quant(exists(e, Ev(e), Vote(e,x)))).

The Discourse Manager is responsible for management of discourse entities. The module detects the explicit and hidden entities, concepts in the sentences, and resolves the different references. The module uses the method of discourse copying. The output of the KQML module is the function call to specific system services. The call contains a parameter list containing among others the sender, the operation and the parameters. The call is executed by the executor engine.

A new approach is presented in (Strassel et al. 2010) related to the DARPA's Machine Reading project. The kernel NLP engine of the project is called FAUST (Flexible Acquisition and Understanding System for Text). The approaches emphasizes the role of semantic, the domain ontology needed to understand the text. The knowledge base is stored as a collection of first-order formulas describing domain specific rules. The engine contains a powerful reasoning engine to inference specific sentence-level interpretation. The engine uses the Stanford NLP engine to perform the linguistic analysis.

DOI: 10.14750/ME.2013.009

Some standard NLP engines are available on the Internet, the most widely used are OpenNLP, Natural Language Toolkit (NLTK), Stanford NLP and Unitex. The Stanford NLP engine (StanfordNLP 2013) provides a general framework for text processing. It includes the following modules: tokenization, part-of-speech tagging, named entity recognition, parsing, and co-reference. The project improved the language engine of the Unitex project (Paumier et al. 2010). The Unitex engine is a general text processing framework containing the following modules: preprocessing, tokenization, dictionary-based morphology analysis.

The importance of optimal architecture and IT techniques is explained well in (Paumier et al. 2010). The paper describes a successful collaboration between academic and industry partners on development of a NLP engine for iPhone devices. The performed internal optimization resulted in a significant speed up of processing. The average time spent to process a sentence dropped from 811 ms down to 15 ms (Paumier et al. 2010).

2.1 Brief history of NLP systems

The history of natural language processing has started around 1950 when Alan Turing has published his paper called "Computing Machinery and Intelligence" (Turing 1950) proposing the criterion of intelligence. Now, it is called Turing test, where a human judge makes real-time written conversation with a human and a computer. The computer program can pass the test if the human judge cannot distinguish between the application and the human.

Machine translation was the first natural language related application, some early projects can be found already some years before 1950 in (Hutchins 2005). Systems simply used dictionary-lookup for searching words, for translation them or for reordering to satisfy the requirements of target languages. These systems generally produced poor results which led to define a more adequate theory of language.

The authors of "Georgetown experiment" in 1954 declared that the machine translation would be solved within a half decade after they have implemented the automatic translation of more than sixty Russian sentences into English (Hutchins 2004).

In 1957 Chomsky published his paper titled "Syntactic Structures" (Chomsky 1957) in which the idea of generative grammar was documented. It results some gain into the field of machine translation. Meanwhile other NLP application areas emerged like speech recognition.

Researchers became more optimistic after the extremely well results of SHRDLU (Winograd 1972) which was a natural language system working in "blocks worlds" with limited vocabularies in 1960's. The next application of NLP systems was ELIZA (Weizenbaum 1966) written by Joseph Weizenbaum between 1964 and 1966. ELIZA has simulated a psychotherapist using almost no information about thoughts or emotions but providing interesting human-like interactions.

DOI: 10.14750/ME.2013.009

The statement that the machine translation will be solved within three or five years, was overestimated and the ALPAC report (J. R. Pierce 1966) in 1966 led to make funds dramatically reduced since the results stayed under expectations after ten years long research.

In the 70's "conceptual ontologies" has begun to be written by many programmers. Ontologies transform real-world information into computer-understandable data like in: MARGIE (Shank 1975) SAM (Cullingford 1981), PAM (Wilensky, 1978), TaleSpin (Meehan, 1976), QUALM (Lehnert, 1977), Politics (Carbonell, 1979), Plot Units (Lehnert 1981). In this time many chatterbots have been developed like PARRY, Racter or Jabberwacky. LUNAR was developed as an interface to a database which contains the lunar rock samples. In the late 70's the semantic issues, communicative goals and plans and discourse phenomena came to the front.

From the end of 1980's the increasing of computational power and the less expensive cost led to the usage of statistical models in NLP with more interest.

2.2 Levels of natural language processing

A natural language processing system can be ranked by the number of levels of the language it utilizes. Levels give a synchronic model of the language contrary to an earlier sequential model which supposes that the levels of language processing follow each other sequentially. Researches recommend regarding levels more dynamic which can interact in variety of orders. In the following the description of levels is presented sequentially but the meaning is essentially conveyed by each level of language.

2.2.1 Phonology level

In phonology level speech sounds within and across words are examined. Three types of rules can be defined in this level:

- **Phonetic rules:** used for sounds within words
- **Phonemic rules:** used for variations of pronunciation of spoken words together
- **Prosodic rules:** used for determining intonation across a sentence and fluctuation in stress.

Phonological analysis can be performed in such NLP systems where the input is the spoken sentences and sound waves are encoded into a digitized signal like a written text form.

2.2.2 Morphology level

Words are composed of morphemes that are the smallest units of meaning. The aim of morphological level is to classify input words into separate morphemes. The meanings of morphemes are not changed across words, therefore an unknown word can be broken into constituent morphemes to determine and to understand the meaning of it. There are many kinds of languages where the morphological analysis differs. E.g. in English language the words are mostly in their original form, they are inflected only in some cases: in plural form of

DOI: 10.14750/ME.2013.009

nouns (+s), past tense of verbs (+ed), etc. In agglutinative languages like Hungarian, the situation is more difficult, the meaning is manifested by the inflection of words instead of e.g. the place in word order. Hungarian words can have prefixes (like in verbs), and many kinds of suffixes (inflection, ending) which determines the meaning of words. The key piece of meaning is the proper specification of stems and suffixes.

2.2.3 Lexical level

In lexical level the meaning of individual words should be interpreted. It can be performed in several ways like assigning a single part-of-speech tag to each word. If a word can only have one possible sense or meaning it can be replaced by its semantic representation. In that case if more part-of-speech tags are determined for a word, the context should choose which one may be used.

Lexicons can be used in lexical level which can be quite simple or arbitrarily difficult. In simple representations a lexicon contains pairs comprising words and their part-of-speech tags, whilst in more difficult cases a lexicon can store semantic classes, limitations on semantic arguments, etc.

2.2.4 Syntactic level

The aim of syntactic level is to analyze words in a sentence and to uncover the grammatical structure of the sentence. The analysis needs the existence of a grammar and also a parser. The output of syntactic level is a kind of representation of the sentence which denotes the dependency relationships between words. Sentences can be fully parsed or can only phrasal and clausal dependencies are specified depending on nature of NLP application. In such languages where word order has important role, the syntax conveys the meaning. In languages with free word order the meaning is influenced mainly by morphological, lexical and semantic levels.

2.2.5 Semantic level

Semantic level focuses on the possible meanings of a sentence considering the interactions among word-level meaning in the sentence. Resolving semantic ambiguous words is an important task of semantic level processing where a word has multiple senses similarly to syntactic disambiguation in syntactic level where a word had multiple part-of-speech. Semantic disambiguation allows only one sense of polysemous words which will be included into semantic representation of the sentence. The disambiguation process needs the remaining part of the sentence to determine which sense of a word should be chosen. Several methods can be found which solves disambiguation tasks, some of them require frequency information for each sense in a corpus, some requires consideration of local context, others uses pragmatic knowledge of the domain.

DOI: 10.14750/ME.2013.009

2.2.6 Discourse level

Levels until discourse level work on words of sentences, it can also be called as sentence-level processing. The discourse level works on longer units. It does not mean that more sentences will be interpreted like they would be concatenated, but the sentences as a whole should be analyzed. The two of the most common processing types in discourse level are the anaphora resolution and the discourse structure recognition. Anaphora resolution replaces the semantically vacant words like pronouns with the belonging entity to which they refer. Discourse structure recognition categorizes sentences of the text into small pieces of which sentences belong to the same discourse component. For example a book can be deconstructed into discourse components such as: preface, acknowledgements, introduction, chapters, etc.

2.2.7 Pragmatic level

The pragmatic level is more complex and work-intensive than all other levels. Pragmatics is often thought as the underlying meaning of the text that depends on such knowledge about world which comes from outside the document. A computer needs to have the same knowledge like people have, about the world to be able to understand pragmatic, but computers can do it more with more difficulty than people can. Information retrieval researchers think that the only way to add pragmatic level into NLP system is to gather all knowledge of a world to use as a reference guide or knowledge base in information systems. The problem with this concept is that building such a huge knowledge takes very long time, it is very costly and looks only the existing knowledge regardless of new information.

2.2.8 Summary of levels

Mainly the lower levels of processing are implemented in current NLP systems. Most applications do not require the implementation of higher levels, at the same time lower levels are more thoroughly researched and implemented. Lower levels work with smaller units like morphemes, words and sentences which mostly rule-governed contrary to regularity-governed higher levels dealing with text and world knowledge. Lower levels of analysis use statistical approaches, whilst symbolic approaches can be used in all levels, although higher levels are implemented in NLP systems very rarely.

2.3 NLP tasks and frameworks

There are numerous tasks in the field of NLP subset of which frameworks usually implemented. Some of the tasks have real applications while others are only building blocks in larger units. The most relevant tasks in NLP researches are

- Automatic summarization
- Co-reference resolution
- Discourse analysis
- Machine translation
- Morphological segmentation

DOI: 10.14750/ME.2013.009

- Named entity recognition
- Natural language generation
- Natural language understanding
- Part-of-speech (POS) tagging
- Parsing
- Question answering
- Sentence breaking a.k.a sentence boundary disambiguation
- Speech recognition
- Speech segmentation
- Topic segmentation
- Word segmentation
- Word sense disambiguation.

Most of the previous tasks are realized by famous NLP frameworks like NLTK (NLTK 2012), Apache UIMA (Apache 2013) or Stanford NLP (StanfordNLP 2013).

NLTK can be used for developing Python programs to process human language data. It provides many easy-to-use interfaces over 50 corpora and lexical resources like WordNet. NLTK implements classification, tokenization, stemming, tagging, parsing and semantic reasoning libraries. NLTK supports mostly English language but tasks of it can be adapted for other languages as well.

Stanford NLP (SNLP) framework has been developed by Stanford Natural Language Processing Group (SNLPG) before many years. SNLP consists of Java-based statistical NLP toolkits for various major computational linguistic problems. Stanford CoreNLP is part of SNLP which is an integrated suite of NLP tools for English which includes: tokenization, POS tagging, NER, parsing and co-reference resolution. Besides CoreNLP, SNLP has individual NLP modules which also provide above tasks. SNLP supports mostly English, Arabic, Chinese, French and German languages. Adaptation of Stanford Parser is available for Hungarian language in *magyarlanc* project (Zsibrita, Vincze, and Farkas 2013).

Apache UIMA is the most robust framework, among others, components of it are also available for both Java and C++. In UIMA annotator components should be configured and pipelined which do the actual work of analyzing unstructured information. Own annotators can be implemented or existing ones can also be extended. UIMA analysis engine provides the following NLP tasks: language identification, tokenization, POS annotation, shallow parsing and named entity recognition.

2.4 Related works

One of the best-known results of Hungarian NLP is the free iSpell dictionary with Hunspell (Németh 2011) spell checker and morphology analyzer which provides accurate spell checking for OpenOffice, Mozilla Firefox browser, Thunderbird email client, Google Chrome,

DOI: 10.14750/ME.2013.009

Internet Explorer, Opera browsers and for users of many other applications. A Hungarian lexical database and morphology grammar is presented by (Trón et al. 2006).

Hungarian Natural Language Processing Group (HNLPG) has also many related researches belonging to the area of NLP. They developed an extension (Szarvas, Farkas, and Kocsor 2006) to the MALLET (McCallum 2002) conditional random field (CRF) named entity recognizer (NER) which contains a parametrisable feature extractor package and adapters for UIMA framework (Apache 2013). Magyarlanc (Zsibrita, Vincze, and Farkas 2013) is a toolkit for linguistic processing of Hungarian language. It contains a sentence splitter and tokenizer which are the extensions of MorphAdoprner (MorphAdorner 2009), a POS tagger and lemmatizer which uses a modified version of Standford POS tagger (StanfordNLP 2013), stopword filtering and dependency parser which is a version of Bohnet parser (Bohnet and Niver 2012) adapted to Hungarian.

The main achievements of HNLPG are the construction of Szeged TreeBank (Csendes et al. 2005), a dictionary which contains syntactic analysis and annotation of sentences and the Hungarian Ontology a.k.a Hungarian WordNet (Alexin et al. 2006) which is a natural language concept set on the basis of WordNet.

2.5 Conclusions

The analysis of existing frameworks shows that main tasks of NLP are properly implemented and frameworks can also be adapted for several languages like Hungarian. HNLPG decided to adapt some NLP tasks such as tokenization, parsing to fulfill the requirement without implementing whole modules.

In general, the most complex task of investigated NLP frameworks is the parsing or sentence analysis. Most of the frameworks do not continue the processing since it should be the task of applications. In natural language controlling systems a required task could be the mapping of parsing result into a function call.

The main objective of our research is to develop a natural language controlling framework which extends the set of tasks of existing NLP frameworks with the ability of controlling applications with natural language commands and queries.

In order to be able to realize the function mapping, the semantic model of domain knowledge, the function description mode and the algorithms of sentence analysis and function mapping should be developed.

Regarding requirements defined in section 1.2 the NLC framework should also be domain- and language-adaptive. Investigated frameworks support several languages but they do not separate tasks into language-dependent and independent parts. The proposed NLC framework should differentiate modules based on this category and should propose a solution to bind the two groups together. Certainly, the adapted tasks of existing frameworks could also be integrated into the proposed NLC framework.

DOI: 10.14750/ME.2013.009

3 Developing a Natural Language Controlling Framework

First goal of our project is to define the architecture of a natural language controlling framework which satisfies the requirements in section 1.2. The processing workflow is more complex than it could be handled in a single module. The complexity of software engineering can be managed with the usage of layered design approach. One of the first proposals in this field is the paper of Goldstein and Bobrow (Goldstein and Bobrow 1980). They argue that a layered modeling structure is more suitable to represent an evolving design. The model uses a network to describe the design process, where the nodes represent the modules and procedures. The layered architecture provides a more efficient management of versions, isolated modular design, implementation and testing and larger scale reusability. The layered architecture can be used to develop flexible applications that are decomposed into subtasks (Sharma, Jalote, and Trivedi 2005).

The logical structure of the proposed NLP system is presented in Figure 3.1. In next sections the requirements and tasks of main modules will be discussed in detail regarding to domain- and language-adaptivity.

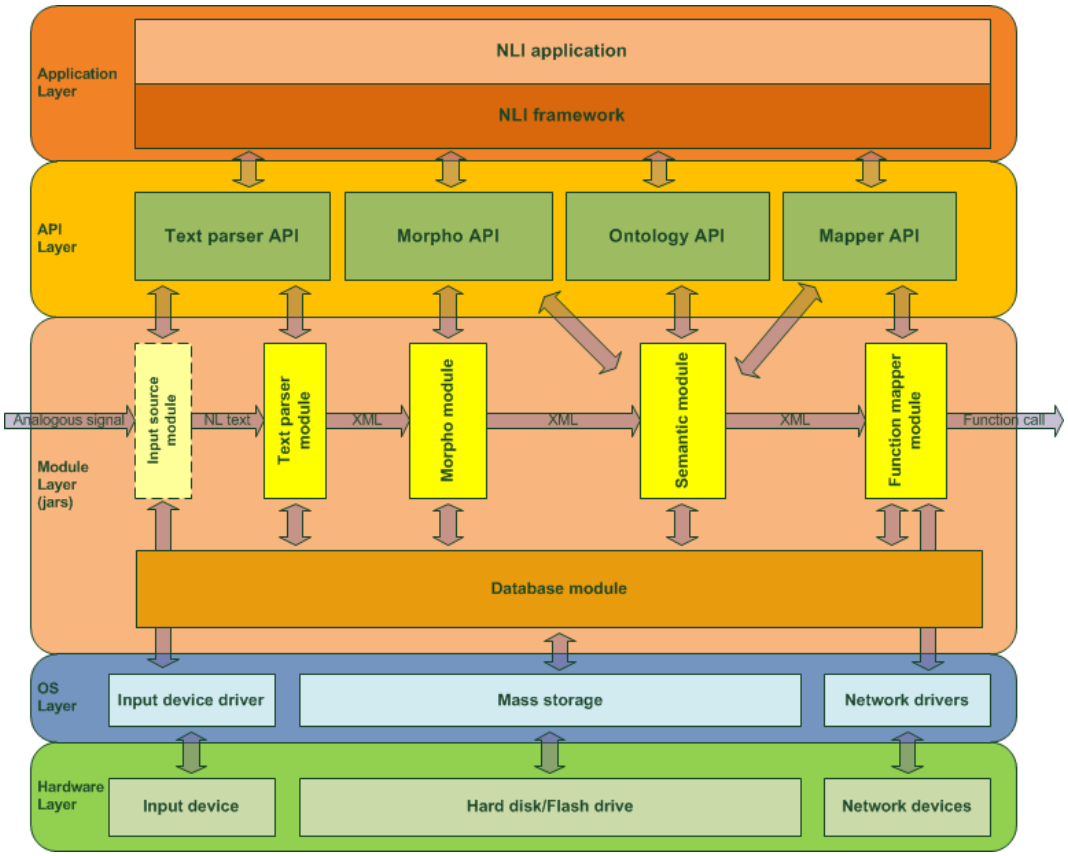

Figure 3.1. *Logical architecture of NLP System*

DOI: 10.14750/ME.2013.009

The requirements of the proposed natural language controlling framework are the following *(R1)*

1. **Domain-adaptivity:** the ability to easily learn concepts and relations of different domains without modifying the inner structure and workflows of framework. The domain knowledge should be loaded into the ontology module through its application programming interface (API). The API layer also provides access to semantic information for framework layer which hides the realization details from developers and users.
2. **Language-adaptivity:** the ability to parse natural language sentences in different languages with only teaching language-dependent parts of framework. Since the modules can be splitted into language-dependent and –independent modules, only text parser and the morphology module should be re-implemented for different-languages. The belonging API provides well-defined interfaces for implementing language-dependent parts of modules.
3. **Extensibility:** the ability to extend the set of functions which wanted to be called by natural language commands. The extension can also involve the extension of domain knowledge when new concepts are defined for new functions. Ontology and function mapper API provides methods which satisfy the current requirement.
4. **Open interface:** the ability to reuse existing components of NLP engines and to implement, refine any part of framework for own needs. There are several NLP engines in the market which have proper solutions for subtasks of frameworks like spell checking, morphology analysis, etc. The task of the developer is to integrate these services into his own implementation using the suggested interfaces which are provided by the API layer of the framework.
5. **Modularity:** the ability to decompose related tasks into independent but connected modules. Modular architecture also provides the reusability and exchangeability of implementations of individual tasks.

As can be seen in Figure 3.1 the natural language controlling can be regarded as a transformation of a text given in natural language into an application function call. It can be formalized as follows:

$$f: T \rightarrow A, \tag{3.1}$$

where

- T : is the set of sentences in natural language
- A : is the set of application functions.

Natural language texts can be composed from sentences which are built up from sequence of words whilst words are collated with using a finite sequence of symbols over a given alphabet. The rules which define how the sentences can be formed using words are called syntax or grammar. Formally,

DOI: 10.14750/ME.2013.009

$$\begin{gathered} w = c_1 c_2 \cdot \ldots \cdot c_n,\ c_i \in \Sigma, \\ S = \{w\}, w \in W \subseteq \Sigma^*, \\ T = \{s\}, s \in S \subseteq W^*, \end{gathered} \tag{3.2}$$

where

- c : is a character of alphabet
- Σ : is the finite character set of the language
- Σ^* : is the set of all words over the alphabet
- W : is a given word
- W : is the finite sequence of words
- W^* : is the set of all sentences over the language
- S : is the set of sentences.

The application functions defined in (3.1) can be decomposed into sets of operations belonging to individual modules. Each module has its own tasks which transforms the output of the previous module to produce the proper output. The composition of (3.1) is given with

$$\begin{gathered} f(T) = \phi_1(T) \circ \phi_2(o_1) \circ \ldots \circ \phi_n(o_{n-1}), \\ o_i = \phi_i(o_{i-1}),\ 1 \leq\ i \leq n, \end{gathered} \tag{3.3}$$

where

- ϕ : is a module function
- o_i : is the output of the i[th] module.

The composition has the following properties:

- associative: $\phi_1 \circ (\phi_2 \circ \phi_3) = (\phi_1 \circ \phi_2) \circ \phi_3$,
- not commutate with each other: $\phi_1 \circ \phi_2 \neq \phi_2 \circ \phi_1$.

Combining (3.1) and (3.3) the next constraints are imposed:

$$\begin{gathered} \phi_1 = \phi(T), \\ \phi_n : \phi_{n-1} \to A. \end{gathered} \tag{3.4}$$

There are four modules defined in NLC framework. Each module has a set of functions the composition of which gives the results or output of modules. The defined modules are

- text parser ($\phi_1 = \alpha$)
- morphology ($\phi_2 = \beta$)
- ontology ($\phi_3 = \gamma$)
- function mapper ($\phi_4 = \delta$).

The starting point of processing in the NLC framework is the reception of a textual input in a specified language. The production of input is not part of the project but it can affect the later

DOI: 10.14750/ME.2013.009

processing like spell checking. Thus the join of input source module ($\phi_0 = \iota$) to the proposed framework should also be analyzed to show the effects and roles in NLC processes.

Consecutive sections describe the requirements and the tasks of each module regarding the effect for making module output.

3.1 Input source module

The input source module is responsible for transforming the human input, which can come from various devices, into text. The input device can be a speech recognizer, a scanner, a handwriting recognizer, a keyboard input or any other equipment which can produce textual output. The common feature of these devices is that they should generate digitized text from some analogous signal where the conversion can have some error rate. The text generation can be denoted formally

$$\begin{gathered} \iota: \Gamma \to T, \\ T = s_1 s_2 \cdot \ldots \cdot s_n, \end{gathered} \tag{3.5}$$

where

- Γ : is the analogous input
- T : is the converted text
- s_i : is the i^{th} sentence of text.

The error rate can be counted from misrecognized parts of the text (sentence or the word). The reference sentence of a sequence is described in (3.6). Generally the error can be written as

$$\begin{gathered} T' = s'_1 s'_2 \cdot \ldots \cdot s'_n, \\ |T| = |T'|, |s_i| = |s'_i|, w_{ij} \in s_i, w'_{ij} \in s'_i, i \leq |T|, j \leq |s_i|, \end{gathered} \tag{3.6}$$

$$\Delta = \frac{\left|\{w_{ij} |\, w'_{ij} = w_{ij}\}\right|}{\left|\{w_{ij}\}\right|}, \tag{3.7}$$

where

- Δ: the error rate of conversion
- w: the number of all words
- T' : the reference conversion which contains the correct sequences of sentences.

Different types of input devices have special mistakes. For example in case of keyboard input, the mistyping gives the faults in input text, at the same time in case of handwriting the misrecognition of similar shapes gives the error in input. The speech recognition is much more difficult than previous methods. The error correction algorithms may depend on the input type to achieve the most accurate output with highest efficiency. A spell checker is usually used in those systems where text cleaning processes are needed. Table 3.1 compares the most frequent input types by errors, algorithms, accuracy and usage convenience.

DOI: 10.14750/ME.2013.009

Table 3.1. *Comparison of input types*

	Typing	OCR	Handwriting	Speech
Device	Keyboard	Scanner, camera	Tablet, Pen	Microphone
Source of errors	Human mistyping	Recognition error, text source mistakes	Recognition error, handwriting mistakes	Recognition error
Used algorithms	-	Walsh Transformation Projection Histogram Zoning	Neural Networks, Genetic Algorithms Greedy Point Match	HMM, DTW
Correction algorithms	Edit distance (Levenshtein, Damerau)	Edit distance, Bayesian methods	Phrase-based models Language models	FST, Word Graphs, Phonetic distance-based methods
Best recognition, conversion accuracy	100%	70%-98%	70%-95%	50%-80%
Convenience of device type for NLI engine in the scale of 1-4	2	1	3	4

3.2 Text parser module

The text parser module gives the interface of NLP system towards the users. The natural language input can be of several kinds, like text, speech, handwriting, etc. Each type of input has its own characteristic; the commonality is that they can be transformed into text.

The main goal of the text parser module is to recognize the building blocks of the incoming text and produce as accurate output for later modules as possible. Considering primary aims the requirements of the text parser module can be summarized as follows

- **Extendibility:** text parsing is far language-dependent, since each language can have its own alphabet, vocabulary. Text parser module should be able to be extended with customized implementations for specific languages.
- **Low response time:** the framework should work in real-time environment thus every algorithm and task should run with using the least execution time.

DOI: 10.14750/ME.2013.009

- **Low memory cost:** the module should use such solutions for algorithm implementation which uses as low memory as possible providing optimal response time.
- **Support Hungarian language:** Hungarian language is an agglutinative language. This kind of languages is usually not supported by famous NLP engines. The framework should provide a solution for the implementation of related operations.

Accordingly, the text parser module should implement the following tasks:

- **Sentence tokenization:** the input text is a single unit which has to be decomposed into sentences. The sentence tokenization is not a trivial problem since the sentence boundaries, specifically 'dot' is overloaded and a solution is needed to be found which decides which one of them is a sentence delimiter or which is not.
- **Word tokenization:** after sentence terminations are detected, words of each sentence should be highlighted.
- **Spell checking:** the mistyped words should be recognized and the corrected version should be provided via spell checking. It has to be executed word by word. A misspelled word can have more possible corrected alternatives in which cases context-sensitive approaches can lead to a success.
- **Named entity recognition (NER):** there are numerous entities in a language which are special like numbers, dates, countries, cities, companies, etc. These data should be handled in a more different way than normal words. Named entities should be labeled with the type of the entity to make later analysis more efficient.

The text parser module has been composed from four operations described in (3.8):

$$\alpha = \pi_{sentence} \circ \pi_{word} \circ \pi_{spell} \circ \pi_{ner}, \tag{3.8}$$

where:

- $\pi_{sentence}$: is the operation of sentence tokenization
- π_{word}: is the operation of word tokenization
- π_{spell}: is the spell checking operation
- π_{ner}: is the named entity recognition process.

The order of operation is defined in (3.8), the order of their execution cannot be changed or reordered.

3.2.1 Sentence and word tokenization

In order to be able to process NL input the text has to be parsed first, then splitted into sentences and the sentences into words. The parsing mechanism can be quite complex, since the '.' Boundary. There are existing segmenters in famous NLP engines (NLTK 2012), (StanfordNLP 2013), (Apache 2013), (MorphAdorner 2009) which usually do not support Hungarian language. Most of these solutions can be adapted for our language, like the

DOI: 10.14750/ME.2013.009

Hungarian Natural Language Processing group has done with Morphadorner's segmenter (Kumar 2009) in "magyarlanc" project (Zsibrita, Vincze, and Farkas 2013).

3.2.2 Spell checking

Due to the most accurate selection of application function in the end of natural language controlling process, the NLC modules individually should provide as accurate results as possible. It is increasingly applied to text parser module, since it produces the base elements of the natural language input. If words of sentences are mistyped, the morphology module will not result the correct analysis, the belonging concepts will not be found and the application function execution will not be done. In order to avoid this bubbling effect a spell checker has to be used which corrects the mistyped/misrecognized words.

A spell checker can work in two ways:

- **automatic:** replace the incorrect word automatically with the correct one, which has the highest goodness value from a list of words
- **manual:** generates a list of possible correct words and the user has to choose one from this list.

In a user-friendly NLC system, the spell checker should work automatically without asking users to decide which alternative is correct for a word. The manual checking could decrease the usability. Spell checking should only work with using dictionaries therefore a new notation,

$$D \subseteq W, \tag{3.9}$$

has to be introduced where

- D: is a dictionary containing words of the language.

The spell checking process is formalized in (3.10)

$$\pi_{spell}: W \to D. \tag{3.10}$$

The spell checking should basically satisfy the following rules:

$$\begin{aligned} &\forall w \notin D: \pi_{spell}(w) \neq \emptyset, \\ &\forall w \in D: \pi_{spell}(w) = w. \end{aligned} \tag{3.11}$$

Formula (3.11) ensures that the output of parser will contain only such words that are in a known vocabulary of the language. There are such situations when it is not enough to fulfill previous rules. E.g. a speech recognizer produces the output

"By a couple of tea for me, please."

Here can be seen that every word is correct, but the word 'by' is not yet in correct form. The result of correction should be 'buy', but it can only be recognized from the context. Another

DOI: 10.14750/ME.2013.009

reason for applying a context-sensitive spell checker is to choose the correct alternative from the set of words. Here we could define some similarity function but combining with the context the result could be made more accurate. The realization technique of determining the probability value is not part of this project, many solutions can be found like Bayesian classifier, clustering-based methods, etc. which can be applied.

When the context of a sentence is determined the algorithm should check that the word is a part of that context or not. If not, the most similar word should be found for replacement of the misrecognized word in the natural language input.

3.2.3 Named entity recognition

Named entity recognition is also a main and important task of NLP engines. There are special words in each language like numbers, dates, companies, addresses, etc. which should be handled in a way that differs from that of normal words. Subsequent modules should not process them fully; they can only use the label which NER has glued to it. Formally, the NER makes the following transformation

$$\begin{gathered} L = \{l_1, l_2, \dots, l_n\}, \\ \pi_{ner}: D \to \wp(L), \end{gathered} \tag{3.12}$$

where L is the set of labels of entities like NUMBER, DATE, COMPANY, etc.

Since different domains can contain different entities, the extendibility requirement of NER task should also be satisfied. The API layer of the text parser module has to provide a solution for customizing used NER instances.

3.3 Summary of text parser module

Text parser module is responsible for receiving natural language sentences, tokenizing them into sequences of words, spelling false words and labeling the special words as named entities. The output of the module can be summarized using previous statements as follows

$$\begin{gathered} \alpha: T \to \{s'\}, \\ s' = \{(w, l)\}, w \in D, l \in L. \end{gathered} \tag{3.13}$$

3.4 Morphology module

Morphology is a basic part of natural language processing. It is relevant increasingly in case of agglutinative languages, like Hungarian. In such languages like English, words are transformed only in a few cases: plural, third person singular verbs, past tense. There are some exceptional plural nouns also. After all, English texts can be analyzed and processed much easier than Hungarian texts where the meaning of a word depends highly on the suffixes. The detection of base forms (stems) of words in a sentence is crucial to determine the meaning of a sentence.

DOI: 10.14750/ME.2013.009

Morphology module should provide two main tasks in NLC system:

- **Morphology analysis:** words of incoming sentences should not only be stemmed but the transformation rules should also be recognized. This task has high relevance among operations of NLC system.
- **Inflection:** it is the inverse process of analysis. A stem of a word can be inflected with using a set of rules. It is relevant when NLC system wants to generate answers in a natural language.

Morphology module in NLC systems has special roles: it represents the border between language-dependent and language-independent modules. To ensure that the later modules do not depend on languages the following requirements should be satisfied:

- **Providing code system for stems:** a concept can be represented with one or more stems in written form. In concept level it is no matter how a concept is written in different languages, the meaning is the same. Therefore a code system should be built up in the morphology module consisting of code-stem pairs. Adding a new language support means inserting stem values for existing codes into the stem-code dictionary. Later modules should use only the codes instead of language-dependent stems.
- **Providing code system for POS values:** since POS values can also be used by later modules, they have to be standardized. Each supported language should use the same POS codes stored in a POS-code dictionary in the result of analysis.
- **Providing code system for suffix/inflection rule values**: in morphology analysis of a word suffixes with inflection rules are defined besides POS values. Later modules use the inflection rules also, so the same inflection rule notation should be used among different languages.

Morphology module receives the output of the text parser module and makes morphology analysis of each word in the incoming sentence. Three code dictionaries should be defined in module:

$$X = X_{stem} \cup X_{POS} \cup X_{suff}, \tag{3.14}$$

where

- X_{stem} : set of codes belonging to stems
- X_{POS} : set of POS codes
- X_{suff} : set of suffix codes.

The analysis process is denoted as

$$\begin{gathered} \beta: \{s'\} \to \{(\omega_{stem}, \{(\omega_{pos}, A\}), \}, \\ \omega_{stem} \in X_{stem}, \omega_{pos} \in X_{POS}, \\ A \subseteq X_{suff} \times X_{suff} \times \ldots \times X_{suff}. \end{gathered} \tag{3.15}$$

DOI: 10.14750/ME.2013.009

Morphology analysis can be a resource intensive and time consuming task, so it is very important to investigate the alternatives for realization considering the real-time property of the NLC system. In next sections the grammar representations and parser methods are analyzed.

3.4.1 Grammar representations

A classic formalism for representing grammar is the Formal Grammar (FG) mechanism. In FG, Σ denotes the set of characters in the language. The symbol Σ^* is for the finite sequence of the characters. The language L is defined as a subset of Σ^*. The language L can be given with a grammar G, where G is a tuple (N, T, P, S) with

- T : the set of terminal symbols from Σ
- N : the set of new non-terminal symbols
- S : the symbol for a sentence
- P : the set of production rules.

The language $L(G)$ denotes the set of sequences from Σ^* that can be derived from S using P. Based on the complexity of the rule-set, Chomsky (Chomsky 1965) has defined four base classes. These four classes are defined as follows: regular grammar with the simplest rules; context-free grammars; context-dependent grammar and recursively enumerable grammars. A widely investigated problem is the role of natural languages within the Chomsky classification. Chomsky has argued (Chomsky 1963) that NL has context dependent characteristics. Others, like (Gildea and Jurafsky 1995) consider NL as a subclass of regular languages as every NL is a finite language, thus finite regular automata can be applied to parse the sentences. One practical drawback of the regular approach is that the related regular automaton is too huge for implementation.

The most widely used representation formalisms for formal grammars are the finite deterministic automata (FSA), the stack automata and the TAG architecture. In the case of FSA, every node of the automata corresponds to a non-terminal element of the grammar, while the directed edges are assigned to the terminal symbols (Manning and Schütze 1999). In the case of stack automata, the automaton has a special memory to save the previous states. The TAG (Tree Adjoining Grammar) uses a hierarchy to store the grammar description. The TAG hierarchy is given with a tuple (N,T,I,A) where T denotes the terminal symbols, N is for the non-terminal symbols, I is a finite set of initial trees and A is the set of auxiliary trees. The leaves of the trees are either terminal symbols or non-terminal symbols that can be substituted by another auxiliary tree. The trees can be adjoined via the substitution leaves.

The cognitive linguistics appeared first in the 1970's. One of the main pioneers of this approach is Langacker (Krenn and Samuelsson 1997). The cognitive approach assumes that the human language reflects the general cognitive processes of the human mind. Several specific grammars were developed, usually with some stochastic learning mechanism. The Word Grammar (WG) proposed by Hudson (Hudson 2007) belongs to this category. WG

DOI: 10.14750/ME.2013.009

represents the grammar with a knowledge graph including all four levels of the human language, namely the semantic level, the syntax level, the morphology and the phonology.

A different approach is implemented in the Dependency Grammar (DG) proposed by Tesniere (Tesniere 1959). The DG uses dependency description between the words of a sentence. The dependency has a head (verbs) and some dependents. The dependency relation corresponds to grammatical functions. The dependency relationship is described with a dependency tree called stemma. The stemma is well-formed if (Robinson 1970)

- One and only one element is independent
- All others depend directly on some element
- No elements depend directly on more than one
- If A depends directly on B and some element C intervenes between them (in the linear order of the string), then C depends directly on A or B or some other intervening element.

A similar formalism is used in the Link Grammar (LG). The link grammar uses only binary relationships, i.e. complex relationships within a sentence are represented with relationships between two words of the sentence.

3.4.2 CL-based parsers

The inflection transformation has a very complex form. In our approach the rule is given with a set of distinct basic transformation rules. An atomic rule corresponds to an unambiguous simple conversion rule. Here are some examples for atomic rules.

- **(#,t)* : a character ’t’ is appended to the end of the word
- **ab(i,o)*(a,át)* : the first occurrence of ’abi’ is replaced with ’abo’ and the ending ’a’ is replaced with ’át’.

In natural languages, the transformation rule depends on the base word. Thus, the inflection transformation can be considered as a classification problem, where the base word is assigned to the best matching transformation class.

In the literature, inflection is usually controlled by a production rule system where the dominating solution is the application of FST or HMM (Manning and Schütze 1999) methods. In our project, a different approach was tested, namely the toolset of Formal Concept Analysis (FCA) (Manning and Schütze 1999). The output of the FSA process is a lattice of formal concepts. This lattice represents the discovered concepts and the generalization and specialization relationships among the concepts. With application of special class label attributes, the concept lattice can be used as a classification method. The input for the FCA analysis is a formal concept defined with $K(O_K,A_K,I_K)$ triplet where

- O_K : set of objects
- A_K : set of attributes
- I_K : a binary relationship between objects and attributes.

DOI: 10.14750/ME.2013.009

Two mapping functions are defined between objects and attributes with

$$h_K(X) = \{a | a \in A_K : \forall o \in X : o I_K a \},$$
$$g_K(Y) = \{o | o \in O_K : \forall a \in Y : o I_K a \}, \tag{3.16}$$

where

$$X \subseteq O_K, Y \subseteq O_K. \tag{3.17}$$

A pair of closed object-set and attribute-set is called formal concept:

$$C(X,Y): h_K(X) = Y, g_K(Y) = X. \tag{3.18}$$

Among the sets of formal concepts a partial ordering can be defined with

$$C_1(X_1, Y_1) \geq_K C_2(X_2, Y_2) \iff X_1 \supseteq X_2. \tag{3.19}$$

A pioneer work on application of concept lattices for classification is the proposal of Zhao and Yao (Zhao and Yao 2006). In their approach, the attribute set of the context is extended with a class label. This label attribute denotes the class membership of the objects. The class label of a node is the aggregation of the class labels in the dominated sub-lattice. A concept in the lattice is consistent if its class label contains only one class. A concept is the most general consistent one if it is consistent but neither of its super concepts is consistent.

In the inflection rule concept lattice, the attributes of the context correspond to the labeled character sequences of words. A label contains positional data on the sequences. The intension of a concept is given by a set of labeled substrings called generalized word. The generalized word at a new concept is constructed with intersection of the corresponding generalized words. A default class value is also defined here as the class with highest support within the dominated concept nodes. According to (Zhao and Yao 2006), a concept lattice can be converted into a decision tree for determining the class attribute from the content attributes. The generated decision tree is a binary rooted tree, where each node is assigned to a generalized word. The classification process at a given node works in the following steps:

1. If the node is consistent, the search terminates and the current transformation rule is applied.
2. If the node is inconsistent, the child nodes are tested.
3. If no child node exists, the default rule is applied; otherwise all child nodes are tested.
4. The test determines a match similarity value for every child. The child with maximum similarity is selected as next target node.

The construction of the classification lattice is based on a corresponding training set. The training set contains samples on inflection rules, like *(labda, labdát)*. In this example, the first word is the base word ‘ball’ and the second is the word in accusative.

The parser module contains, beside the inflection engine, another unit to manage the different suffixes. In the language, there is a relative rigid rule on combination of the different suffixes.

DOI: 10.14750/ME.2013.009

As the order of the components can be described with a regular grammar, a Finite State Automata (FSA) was implemented to control the ordering of the morphemes. The FSA contains a finite set of states where every state here corresponds to a morpheme. There is an edge from morpheme A to morpheme B if AB is grammatical sequence of suffixes.

The third unit in the grammar module is the stem dictionary. The language has a set of valid stems which can be inflected with different suffixes. As this set is a list of static words it can be implemented with a trie (or prefix) structure. The trie structure is a special tree to store words. The words with the same prefix part share the same tree section started at the root. This structure is very suitable for efficient storage and search operations as the same prefix part is stored only once for several words.

3.4.3 Rule-based parsers

In NLP systems the morphology has quite a huge role to process input with higher accuracy. There are two main functions of morphology which generally applied in natural language processing: stemming and inflection. In the following the stemming will be discussed in detail.

There are many rule-based stemmer approaches for English language since the1960's. One of the most popular is the Porter-stemmer (Porter 1980) because of its simplicity and efficiency. Consonants and vowels are distinguished by algorithms, where if a letter is not a consonant then it is a vowel. A consonant is denoted by c, a vowel by v. The list of consonants of length greater than 0 will be denoted by C, and the list of vowels of length greater than 0 will be denoted by V. Using previous notations any word can be formalized with one of the following forms:

$$\begin{matrix} \mathrm{CVCV}\ldots\mathrm{C} \\ \mathrm{CVCV}\ldots\mathrm{V} \\ \mathrm{VCVC}\ldots\mathrm{C} \\ \mathrm{VCVC}\ldots\mathrm{V}. \end{matrix} \tag{3.20}$$

These formulas can be represented by a single form

$$[\mathrm{C}]\mathrm{VCVC}\ldots[\mathrm{V}], \tag{3.21}$$

where *[X]* denotes arbitrary occurrence of its content. The form can be simplified further with using tag $(VC)^m$ which represents the *VC* repeated m times. So the final formula can be written as follows:

$$[\mathrm{C}](\mathrm{VC})^{\mathrm{m}}[\mathrm{V}]. \tag{3.22}$$

The stemming is performed in 5 steps using rewriting rules. Steps have predefined order and each step contains alternative rules. The rules define suffix replacements belonging to a given condition. A rule is denoted by the form

$$(\mathrm{condition})\mathrm{S}_1 \rightarrow \mathrm{S}_2. \tag{3.23}$$

DOI: 10.14750/ME.2013.009

A rule can be applied if the ending of the word fits to $S_{1,}$ and after cutting S_1 off the condition is fulfilled by the remaining stem.

A condition generally can be given in terms of m, e.g.

$$(\mathrm{m} > 1)\mathrm{ság} \rightarrow. \tag{3.24}$$

The condition part can also contain the following

- *S: the stem ends with S
- *v*: the stem contains a vowel
- *d: the stem ends with a double consonant
- *o: the stem ends with cvc, where the second c is not W, X, Y
- expressions: and, or, not.

If more than one rule can be applied then the rule with the longest ending will be the winner. After a successful application of a rule set, the algorithm jumps to the next rule set. If no rule fits in a set the process will continue with next one. After processing 5 rule sets, the algorithm will terminate.

A general affix representation language has also been developed by Porter, called Snowball which can handle prefixes besides suffixes. Using Snowball (Porter 2005) 14 European languages have stemmer including Hungarian. Tordai-stemmer (Tordai and de Rijke 2005) is a Hungarian stemmer based on Snowball (Porter 2005). There are many alternatives of Porter-stemmer like Lovins-stemmer (Lovins 1969), Paice-Husk-stemmer (Pacie 1994) or Krovetz-stemmer (Krovetz 1993).

Since natural languages are usually quite difficult to process, the accuracy of stemmer algorithms cannot be a 100%. Even people can make mistakes for some ambiguous words. Generally, three kinds of mistakes can be distinguished: under-stemming, over-stemming or misconstruction. The last two error types can be reduced with using an exception dictionary.

3.4.4 Summary of morphology module

Implementation of morphology module is quite complex due to the grammar rules of different languages. There is no one common solution which supports all languages in one, thus current module should be planned to be adapted easily for several languages without touching other modules of framework. Hence a common interface with specified language should be implemented and it is developed with two main operations: analysis and inflection. This kind of approach is suitable mainly for agglutinative languages like Hungarian, but with some specific integration analytic and fusional languages can also be fit into the system.

There are many stemmer and parsers of different languages which can be applied in NLC framework. The rule-based approaches are proposed to be integrated because of their simplicity and efficiency which are needed for ensuring real-time processing.

DOI: 10.14750/ME.2013.009

3.5 Ontology module

One of the key components of the NLP engine is the ontology module which takes the morphology output consisting of codes and transforms into sentence analysis tree. Sentence analysis can be achieved only in view of concepts of a given domain. Without such semantic knowledge base, analysis can be made using statistical methods which do not lead to as accurate result as the proposed one. There are several alternatives to store the ontology knowledge base which will be discussed in next sections.

3.5.1 Semantic representations

In the case of semantic networks, concepts are represented as nodes in a graph and the binary semantic relations between the concepts are represented by named and directed edges between the nodes. Semantic network models usually include a graphical representation component too. One of the most important members of this family is the RDF graph model. An RDF graph is a set of triplets, each consisting of a subject, a predicate and an object. Each triplet represents a statement of a relationship (predicate) between the concepts (subject and object) denoted by the nodes that are connected by a directed link (pointing to the object).

Frame-based systems use entities like frames and their properties as a modeling primitive. Value restrictions (facets) can be defined for each attribute (slot), and the values (or fillers) of these slots can either be atomic values or other embedded frames. The notion of frame was originally introduced by Minsky in (Minsky 1975). According to his definition, a frame is a data structure for representing a concept, which can be unique or generic.

First-order predicate logic (FOPL) is a flexible, well-understood and computationally tractable approach to knowledge representation, which uses a wholly unambiguous formal language interpreted by mathematical structures. It is a system of deduction that extends propositional logic by allowing quantification over individuals of a given domain of discourse. Predicates are symbols that refer to the relations that hold among some fixed number of objects in a given domain. Objects are represented by terms, which can be defined as constants, functions or variables. FOPL constants refer to exactly one object, and are conventionally depicted as single capitalized letters.

Description logic (DL) is considered the most important knowledge representation formalism unifying and giving a logical basis to the well-known traditions of semantic networks, frame-based systems, semantic data models and object-oriented representations. It is semantically based on, and hence is a subset of FOPL. The DL syntax (Baader et al. 2003) contains two disjoint alphabets of symbols that are used to denote atomic concepts, designated by unary predicate symbols, and atomic roles, designated by binary predicate symbols; where the latter are used to express relationships between concepts. Terms are then built from the basic symbols using several kinds of constructors.

DOI: 10.14750/ME.2013.009

3.5.2 Tasks of semantic module

Previous modules made syntax processing on natural language input. The output of the morphology module is analyzed further semantically in ontology modules. The goal of the domain ontology model is to describe the knowledge of a chosen domain to help to understand the semantic of incoming sentences. People can easily know which the meaning of words of sentence is, because they have that knowledge which we have to model.

The main flows of the domain ontology module are

- Building the domain ontology model
- Detecting concepts of a sentence
- Analyzing an incoming sentence
- Generate output to the function mapper module.

There are several methods that can help to process a sentence in soft computing. In natural language area statistical methods are not enough in case of such difficult languages like Hungarian. This means that besides syntactic processing, we have to model the domain knowledge first to be able to analyze incoming sentence. The novel semantic model and sentence analyzer algorithm can be found in Chapter 4.

3.5.3 Summary of ontology module

Ontology module has high importance in NLC process since the meaning of incoming sentence tries to be retrieved. Sentence analysis needs apiori knowledge about the domain which is built up in the ontology module. Domain-adaptivity is one of the key requirements of our NLC system so ontology module provides it with corresponding interfaces and methods. Since morphology module ensures language-independency for ontology module, it uses only codes from the code system of the previous unit and therefore basically no new implementation of ontology processing is needed. Certainly, the open interface requirement allows this for developers, but it can be called as a possible but not required property.

3.6 Function mapping module

NLI engine has to be planned in such way that several applications can be built which can accept natural language input and can control different systems. The first step in describing application functions is to collect the possible controllable systems and to categorize them. It is very important to prepare the NLI engine for the demands of lots of possible applications. In the following a short overview can be read about controllable application types and systems and their properties.

3.6.1 Controllable application types

Lots of existing systems can be enumerated here which can be extended with natural language interface. NLI engine has to be prepared for a lot of possible applications which can be

DOI: 10.14750/ME.2013.009

controlled with natural language. The controlling part is very compound and can be analyzed after the main application types will be collected and discussed.

The main application types which can be extended with NLI are: information systems, database systems, web portals, web shops, search engines, navigation systems, expert systems, GUI applications, Windows applications, office applications, VoIP applications, designer applications and robot controlling development systems (IDEs).

In information systems the main task is to query the information after the database is created and filled. Many database systems can be extended with a natural language search interface where the user can enter the question in natural language format and the system answers it with a result of a query.

There are many web portals and web shops where the natural language communication will be very convenient. Instead of many clicks we can find and order a product as we would take it in a real shop. Besides queries data creation, modification and deletion can be needed in this case. For example, if we have a cart in a web shop we can put (create) items into it, we can modify them or remove them from the cart.

In search engines the information retrieval is the most important thing so the query-like sentences are dominant like in navigation systems and expert systems, too. If the database creation will be made through NLI, the DDL, DML operations can also be used.

In GUI applications many functions can be controlled by a natural language. Here the specific function calls, the object manipulations are frequent with using results of queries. E.g. a form designer can accept commands to create form elements, to position them, to modify properties etc. Here the natural language text has to be converted to specific function calls. Besides these, the states of objects have to be stored and can be queried anytime.

Robot controlling is typically that kind of area where users give commands to the robot and it will execute them. Constructing controlling commands in this case depends on the knowledge or capabilities of robots, since we cannot instruct the robot to such thing which cannot be understood by it.

Development systems are one of the most difficult areas because natural language command has to be converted into a source code of a programming language.

In designing function mapper the categorization of controllable systems can be achieved to be able to determine the common function types which have to be implemented in NLI engine.

3.6.2 Controlling sentence types

Human communication consists of a lot of things, like spoken sentences, mimicry, ostentation, etc. In natural language interface design the sentences are the most important compoenents since NLI engine will not be able to analyze the visual information of people (leastways not this version). The sentences can be classified into five categories.

DOI: 10.14750/ME.2013.009

A “declarative sentence” or “declaration”, the most common type, commonly makes a statement. An "interrogative sentence" or “question” is commonly used to request information, but sometimes not like in case of rhetorical questions. An "exclamative sentence" or “exclamation” is generally a more emphatic form of statement expressing emotion. An "imperative sentence" or “command” tells someone to do something. An optative sentence is not as common as the other four sentence types and many appear as fixed sayings. We can use an optative sentence when we want to express a wish, hope or desire.

The different types of sentences can be bound to different operations as commands in computer systems. Commands can be chained where the concepts of a natural language sentence can be bound to one or more commands and the result is generated after the end of processing the command chain.

The proposed model for representation of application function descriptions and the mapping algorithm is discussed in Chapter 4 in detail.

3.6.3 Summary of function mapping

In order to build an NLC framework which fulfills the extendibility requirement in case of application functions, deep investigations are demanded on controllable systems. Several types of applications can be found with the potential of extension with NLC. The common property of these systems that a function description set should be defined and bound to the concrete functions to be called. Hence function descriptions should contain semantic information also, like predicate concepts, constituents, etc. besides programmatic properties like names of functions, parameters, parameter types, methods to be executed, etc. The function mapping module should use the notation of the ontology module in order to satisfy the language-independency and adaptivity as well.

3.7 Summary of results

In this chapter the structure of our NLC framework has been introduced. The new scientific results can be summarized as follows:

Thesis 1.

A novel structure of natural language controlling framework has been developed fulfilling requirements specified in recommendations (R1). The architecture of the framework is based on the developed formal information flow model. The framework contains four modules in a linear structure. The proposed architecture provides domain-adaptivity, language-adaptivity and high extensibility with an open interface. These properties provide high level reusability of the framework in software development in the field of human-machine interfaces.

DOI: 10.14750/ME.2013.009

4 Developing semantic models and algorithms

One of the important stages of our project is to find a proper representation form of domain knowledge by means of which natural language sentences can be analyzed. The aims of current phase are summarized as follows: *(R2)*

1. Defining a semantic model for the representation of domain knowledge extended with part of sentence and morphological information.
2. Constructing a function description model extended with constituent information for the proper mapping of sentence analysis into function description.
3. Developing a sentence analysis algorithm which converts the morphology analysis into a sentence analysis tree.
4. Developing a function mapping algorithm which maps the sentence analysis to the proper function description.

Ontology-based semantic modeling is widely used at higher level of information processing. The term ontology can be defined as an explicit "specification of conceptualization" (Gruber 1995). Set of concepts and terms can be defined by ontology to define and represent the knowledge of domain. The ontology has the following parts:

- Concepts, relationships and other entities which are relevant for modeling a domain
- Definitions of representational vocabulary which provide meaning and constraints on the usage.

The key element of ontology is a methodology that determines what the primitives of the conceptualization are and how to determine these primitives from the domain. Based on (Erdmann 2001), an ontology can be given formally with a quintuple

$$O(C, A, isa, atype, F), \tag{4.1}$$

where

- C: is a set of concepts (classes)
- A: is a set of attributes for concepts in C
- $isa \subseteq C \times C$: is the specialization relationship among concepts
- $atype \subseteq C \times A \times C$: is the type signature of attributes
- F: are formulas in predicate logic to describe the rules of concept management.

The set of logic formulas has two components: the first is the set of universal rules and the second is the domain specific rules. It is widely accepted that the set of universal rules should include

$$\forall c \in C: isa(c, c), \tag{4.2}$$

DOI: 10.14750/ME.2013.009

$$\forall c_1, c_2, c_3 \in C: isa(c_1, c_2), isa(c_2, c_3) \implies isa(c_1, c_3), \tag{4.3}$$

$$\forall c_1, c_2 \in C, a \in A: isa(c_1, c_2), atype(c_2, a, c) \implies atype(c_1, a, c). \tag{4.4}$$

In some ontology models, the ontology contains, beside the concepts, also objects as separate elements that can be given as

- D : a set of objects with a new relation
- $instanceOf \subseteq D \times C$.

In this case, the following universal rule is also part of the rule system:

$$\forall c_1, c_2 \in C, o \in D: isa(c_1, c_2), instanceOf(o, c_1) \implies instanceOf(o, c_2). \tag{4.5}$$

The next step is to modify Erdman's model to satisfy the requirements to make sentence analysis using ontology.

4.1 Modified ontology model

In domain ontology model concepts are the main elements. There are the next concept types in model:

- Abstract concept
- Predicate concept
- Individual concept.

Abstract concept groups those child concepts which belong to the same category. Abstract concept is such a concept which can mostly have abstract or individual concept child elements. Only *ISA* connection may be between abstract and other abstract or individual concepts which mean that the child concept is a specialization of the abstract concept.

Predicate concept is such a concept which describes an action. Predicate concepts can be built up usually from a verb or infinitive. The central element of a sentence is the predicate which can have many connecting concepts such like subject, object, adverbs, etc. A very important part of the sentence analysis is to find the predicate. Other words of sentences can be interpreted when we know how they are connected to the predicate: which can be the subject, object, etc. Model of predicates can limit or filter the possibilities and can speed up analysis. Any other concept types can be connected to a predicate. The most interesting part of a connection is the edge. It stores information about the form or format of connecting concepts.

The individual concept is a concrete concept which has no more special child elements. The individual concept must have a parent abstract concept. The individual concept is connected to the abstract concept with an *ISA* connection.

The edges between the concepts can store very important information also, which refers to the morphological (syntactic) analysis and codes defined in that module. Including morphology info in the domain model the time cost of sentence analysis can be reduced.

DOI: 10.14750/ME.2013.009

The central concept of the sentence analysis is the predicate. Any other part of the sentence is determined after finding the predicate which also specifies the available set of belonging concepts. E.g., in the case of predicate 'eat', we can associate the connected concepts such as food, restaurant, time, etc. In the analysis the goals are

- Identifying the concepts in the predicate-based sentence
- Finding the role of concept in the context of predicate (such as subject, object, adverbs of place, etc…).

In most sentences identified concepts assign the roles directly in which cases the keyword highlighting approach is suitable. It explains those situations when users do not speak a language as a native language, the used inflections, grammars are not accurate, but the meaning can be understood well. At the same time there are such examples where inflections, prefixes and prepositions have high importance in the proper analysis.

Therefore, the knowledge model should contain the following required elements

- Concepts which are assigned by stems of words
- Morphology labels that help to identify the role of the concept in the sentence
- Relationships between concepts which build a hierarchical structure of domain knowledge.

The benefit of using morphology labels is dual:

- It helps to choose from in-sentence roles when more instances of the same concepts are identified in a sentence, like in the following example,

  "Navigate from Miskolc to Budapest!"

  Miskolc and Budapest are equally city concepts which are themselves not enough to determine the in-sentence role. *'From'* and *'to'* prepositions are needed to refine the analysis.

- If a concept is unknown but its morphology label is available, it can help to identify the concept of which instance or child can be the unknown concept based on the in-sentence role.

Regarding previous requirements, the domain knowledge model has properties of ontologies and properties of Minsky's frame-based representation (Minsky 1975) together. The model can be denoted as:

$$\mathcal{M} = (C, R), \tag{4.6}$$

where

- C : is the set of concepts
- R : are the directed relations between concepts.

The concept set can be built up from distinct sets of three kinds of concepts such as

DOI: 10.14750/ME.2013.009

$$C = C_P \cup C_A \cup C_I, \tag{4.7}$$

where

- C_P : set of predicate concepts
- C_A : set of abstract concepts
- C_I : set of individual concepts such as objects.

Relations form a subset of Cartesian product of concepts and the relations have three specialized forms

$$R \subseteq C \times C, \tag{4.8}$$

$$R = R_{pred} \cup \mathrm{R_{isa}} \cup \mathrm{R_{attr}}, \tag{4.9}$$

where

- R : is the set of relations
- R_{pred} : set of predicate relationships
- R_{isa} : set of specialization relationships
- R_{attr} : set of attribute relationships.

The individual relationships can be further constrained with following rules

$$\mathrm{R_{pred}} \subseteq C_P \times \{C_A, C_I\}, \tag{4.10}$$

$$\mathrm{R_{isa}} \subseteq \mathrm{C_A} \times \{C_A, C_I\}, \tag{4.11}$$

$$R_{attr} \subseteq C \times \{C_A, C_I\}. \tag{4.12}$$

A concept has some well-defined properties like slots in the frame-based knowledge representation. A concept can be denoted with the following triplet,

$$c = (n, \Omega, \Psi), \tag{4.13}$$

where

- n : is a unique name which identifies the concept
- Ω : is the list of the parts of speech for the concept
- Ψ : is a set of stems belonging to the concept.

Language-adaptivity and language-dependency of the sentence analysis and the function mapping module can be ensured with defining a set of technical code for stems, POS codes and suffix codes introduced in (3.14).

The language-dependent textual representation can be retrieved with a proper query of the stem code. There are several cases where the predicate concepts are compounded from more parts like “how **far is** …”, “what **color is** …”, etc. which should always be handled together.

DOI: 10.14750/ME.2013.009

Here the description of the general predicate "is" needs to be extended with the related part of the predicate. The stem part of the concept can be denoted with

$$\Psi = \{\psi\} : \psi = (\omega_{stem}, \omega_{ext}), \omega_{stem}, \omega_{ext} \in \mathrm{X_{stem}}, \tag{4.14}$$

where

- ω_{stem} : code of stem
- ω_{ext} : code of extension stem.

Relationships have also to be augmented to be able to treat in-sentence roles during the analysis. We can define five top-level categories of constituents: *predicates*, *subjects*, *objects*, *adverbs* and *adjectives*.

Constituents are denoted with

$$\mathrm{K} = \{pred, subj, obj, adv, adj\}. \tag{4.15}$$

The child concept in a relation can be labeled with the part of sentence information and the morphology label which implies the in-sentence role. The following function can be defined which assigns the previous properties to a related concept pair:

$$h : R \to \{\kappa, \wp(X_{suff}) | \kappa \in \mathrm{K}\} \tag{4.16}$$

where

- r_i: is a relation between concepts
- κ: is the constituent of the child concept.

Applying (4.16) the definition of the attribute connection can be refined as

$$R_{attr} = C \times \{C_A, C_I\} : h(r_i) = \big(adj, \{\omega_{suff}\}\big), \omega_{suff} \in \mathrm{X_{suff}}, r_i \in R_{attr}. \tag{4.17}$$

4.2 Representation of sentence analysis tree

Initially, we have a set of codes belonging to words labeled with morphological information. A word resulted by morphology analysis can be denoted as

$$W = \{\big(\omega_{stem}, \{(\omega_{pos}, \mathrm{A})\}\big)\}, \tag{4.18}$$

where

- ω_{stem}: is the code of a stem
- ω_{pos}: is the POS code of a stem
- A: is the set of inflection analysis.

In agglutinative languages there is a function called inflection which transforms the initial word with a given rule as

$$f_{inf} : (w, \alpha) \to w', \tag{4.19}$$

DOI: 10.14750/ME.2013.009

where

- w: the initial word
- α: the inflection rule
- w': the inflected word.

The morphology analysis contains the list of rules which should be applied for the stem one by one to get the inflected result.

$$\mathrm{A} = \{\alpha_i\}, \tag{4.20}$$

$$w_i = f(w_{i-1}, \alpha_i), \tag{4.21}$$

$$w' = w_1 + w_2 + \cdots + w_n, n = |A|. \tag{4.22}$$

The analysis of a word can result in more alternative solutions in the following cases:

- **homonymy:** where words have different unrelated meanings sharing the same spelling
- **polysemy:** where words have different related meanings sharing the same spelling.

Morphology handles words separately which should provide all possible cases of analysis described in (4.20). Later the semantic module will choose which solution fits to the context.

The goal of sentence analysis is to transform the morphology analysis into a sentence analysis tree.

$$f_{analysis} : W \to S_{tree}, \tag{4.23}$$

$$S_{tree} = (N, \to_s), \tag{4.24}$$

$$N = (C', \kappa), C' \in C, \kappa \in \mathrm{K}, \tag{4.25}$$

$$\to_s \subseteq N \times N, \tag{4.26}$$

where

- S_{tree}: is the sentence analysis tree
- N_s: is the nodes of tree
- $\to_s$: edges between nodes in tree.

Since the sentence analysis should always contain predicate, the following rule should be fulfilled by the sentence analysis tree

$$n_p = (C_P', pred) \in N_s, C_P' \in C_P, \tag{4.27}$$

$$\forall n \in N \Rightarrow n_p \to n \wedge \to \in \to_s . \tag{4.28}$$

The sentence analysis is performed by Algorithm 4.1.

DOI: 10.14750/ME.2013.009

```
Input: morphologically analyzed sequence of words in XML format
Output: sentence analysis tree
concepts = DetectConcepts(sentence)
predicate = FindPredicate(concepts)
interrogative = FindInterrogative(concepts)
if predicate is null then
    predicate = defaultPredicate
predicateConnection = FindPredicateConnection(predicate)
ChildConcepts = { }
AttributeMap = { }
foreach child ∈ ChildrenOf(predicateConnection) do
    foreach c ∈ concepts do
        if c is processed  or c is predicate or c is interrogative or not isChild(child, c) then
            continue
        suffix = SuffixOf(c)
        foreach κ ∈ ConstituentsOf(child) then
            if (suffix ∩ SuffixOf(κ) ≠ ∅) then
                ConstituentOf(c) = κ
                if c not in ChildConcepts then
                    ChildConcepts = ChildConcepts + c
if interrogative is not null then
    iConstituents = FindConstituentsForInterrogative(interrogative)
    if HasSuffix(interrogative) then
        foreach sc ∈ FindConceptToSuffix(predicateConnection, SuffixOf(interrogative)) do
            attributeConnections = FindAttributeConnections(sc)
            foreach a ∈ attributeConnections do
                foreach ac ∈ ChildrenOf(a) do
                    foreach acc ∈ ConstituentsOf(ac) do
                        if iConstituents contains acc then
                            defaultConcept = FindDefaultConcept(sc)
                            ChildConcepts = ChildConcepts + defaultConcept
                            AttributeMap[defaultConcept] = interrogative
    else
        pos = PositionOf(interrogative)
        parent = find concept in concepts where position = pos+1
        if parent is null then
            iChild = FindChildConceptForInterrogative(predicateConnection, interrogative)
            ConstituentOf(iChild) = ConstituentOf(interrogative)
            ChildConcepts = ChildConcepts + iChild
        else
            attributes = FindAttributeConnections(parent)
            if attributes is not empty then
                foreach a ∈ attributes do
                    foreach ac ∈ ChildrenOf(a) do
                        foreach acc ∈ ConstituentsOf(ac) do
                            if iConstituents contains acc then
                                ConstituentOf(interrogative) = acc
                                AttributeMap[parent] = interrogative
            else
                iChild = FindChildConceptForInterrogative(predicateConnection, interrogative)
                ConstituentOf(iChild) = ConstituentOf(interrogative)
                ChildConcepts = ChildConcepts + iChild
BuildTree(ChildConcepts, AttributeMap)
```

Algorithm 4.1. Making sentence analysis tree from morphology analysis

DOI: 10.14750/ME.2013.009

4.3 Representation of application function descriptions

In mathematics a function is defined as a relation between some inputs and a set of output which satisfies that every input is related to only one output.

$$f: X \to Y. \tag{4.29}$$

In application development functions that are also called procedures can also receive inputs such as mathematical functions and do some actions where they are utilized. A procedure is usually does not have output as return value, a function does. Formally the application function can be defined as

$$\begin{gathered} P = \{p_i\}, \\ f: P \times P \times ... \times P \to A, \end{gathered} \tag{4.30}$$

where

- P: the set of parameters
- A: is the set of actions which should be executed using parameters.

The goal of function mapping is to transform sentence analysis into a function description in order to be able to execute the real action related to the description. Two main tasks can be defined to satisfy the function mapping:

1. Find the proper function description to sentence analysis.
2. Map nodes (concepts) of sentence analysis to function parameters.

Function definition should be extended with such information which belongs to the semantic representation to make function mapping possible.

$$f: \{(\{\mathrm{c_P}\}, \mathrm{P})\} \to A, c_P \in C_P, \tag{4.31}$$

where

- c_P: is a predicate concept
- Π: is the set of parameters.

To be able to map nodes of the sentence analysis tree into parameters, description should also be completed with some constituent information like

$$p = (c, \{\kappa\}, f_r), \tag{4.32}$$

where

- c: is the concept belonging to the parameter
- $\{\kappa\}$: is the set of constituents belonging to the parameter
- f_r: is the compulsory function which assigns mandatory value to the parameter.

$$f_r: P \to \{0,1\}. \tag{4.33}$$

DOI: 10.14750/ME.2013.009

Using (4.33) most of the concepts can be mapped and function mapping requirements are satisfied. However, there is a problem which is not handled properly yet with the previous approaches: when more instances of the same concepts can be found in the analysis tree, their mapping is ambiguous. In such situations the concept with its parent concept can assign the belonging function parameter. E.g., there are more adjectives of quality or quantity in the sentence analysis, the marked concept designates the function parameter as can be seen in following example.

"Exchange 20 American dollar to euro!"
("Válts át 20 amerikai dollárt euróra!")

Here we can see only one amount of currency. There is the following function:

```
Exchange(source, target, source_amount, target_amount).
```

Since the same constituent like adjectives of quantity belongs to both amount parameters, only the knowledge of the constituent is not enough to the proper mapping. We should specify for *source_amount* parameter that the belonging concept is the object of the sentence, and for the *target_amount* parameter that the belonging concept is the adverb of result or the adverb of goal of the sentence. The modified formal definition of the function description can be seen in (4.34).

$$\pi = (c, \{(\kappa, \{\kappa_d\})\}, |\{\kappa_d\}| \geq 0), \kappa, \kappa_d \in \mathrm{K} \tag{4.34}$$

where $\{\kappa_d\}$: is the set of dependent constituents.

The function mapping is performed by Algorithm 4.2.

DOI: 10.14750/ME.2013.009

```
Input: sentence analysis tree in XML format
Output: mapped function name with parameter values
Functions = find all function descriptions
Predicate = FindPredicateConcept(tree)
Parameters = { }
Scores = { }
foreach func ∈ Functions do
    score = 0
    if Predicate in predicates of func then
        score = score + PREDICATE_SCORE
    if count(parameters of func) = 0 then
        Scores[func] = score
        Continue
    ParamValues = { }
    foreach p ∈ parameters of func do
        pConcept = ConceptOf(p)
        Concepts = find concepts in tree where concept=pConcept
        foreach c ∈ Concepts do
            foreach κ ∈ ConstituentsOf(p) do
                if κ = ConstituentOf(c) then
                    if hasDependencies(κ) then
                        foreach d ∈ dependencies(κ) do
                            dConcept = find concept in analysis where constituent=d
                            if dConcept is not null then
                                ParamValues[indexOf(p)]=valueOf(dConcept)
                                if p is required then
                                    score = score + REQ_PAR_SCORE
                                else
                                    score = score + OPT_PAR_SCORE
                    else
                        ParamValues[indexOf(p)]=valueOf(c)
                        if p is required then
                            score = score + REQ_PAR_SCORE
                        else
                            score = score + OPT_PAR_SCORE
    Scores[func] = score
    Parameters[func]=ParamValues
winnerFunc = find function where Score[function]=max
rp = find required parameters in Parameters[winnerFunc] where value is null
if rp is not empty then
    q = generate question for rp
    result=q
else
    result=winner function with parameter values
```

Algorithm 4.2. Algorithm of function mapping

4.4 Summary of results

In this chapter the developed knowledge and function description models have been introduced. The new scientific results can be summarized as follows:

DOI: 10.14750/ME.2013.009

Thesis 2.

A novel semantic model is developed which satisfies the requirements of the knowledge representation format in our proposed natural language controlling system (R2). The model is an extension of ECG model proposed by (Varga 2011). The implemented extensions (three new types of nodes, three novel edge types completed with morphology and POS labels) ensure a more efficient knowledge transformation between input sentences of a language and the function call generator module. The novelty of the developed function signature model is the inclusion of POS information for function mapping. A deterministic algorithm has been implemented to convert the sentence analysis tree into API function calls.

DOI: 10.14750/ME.2013.009

5 Optimization in NLC framework

Natural language controlling is a complex problem with many small but resource intensive tasks. In order to be able to apply the framework in industrial area it should satisfy the real-time requirement which means that the delay between text typing and function execution should be tolerable. This requirement can be performed only if time consuming tasks and algorithms of the framework are optimized. Accordingly, optimization should also consider time cost reduction, memory/storage usage and the accuracy of processes, since a system could be unusable if its time cost is optimized but it produces low accuracy.

Building blocks of optimization are described in (5.1).

$$c = \frac{(w_t \cdot c_t + w_r \cdot c_r)}{c_a} \rightarrow min., \tag{5.1}$$

where

- c : total cost of execution [-]
- c_t : time cost of execution [ms]
- w_t : weight of execution cost [1/ms]
- c_r : resource cost (memory/storage) of execution [B]
- wr : weight of resource cost [1/B]
- c_a : accuracy of execution [-].

It can be seen from (5.1) that the time and resource costs are in inverse proportion to the accuracy, which means that the aim is to achieve the highest accuracy besides the lowest time and resource costs.

The goal of this chapter is to specify the optimal algorithms in natural language controlling and to give recommendations for minimizing costs defined in (5.1).

5.1 Optimization problems

Regarding the involved metrics, the optimization problems can be classified into two main groups: single-criterion and multi-criterion problems. The optimization algorithms usually work with real-valued functions, thus they can be applied for single-criterion problems. In the case of multi-criteria optimization, we meet some new problems to be solved (Ghosh and Dehuri 2005). A multi-criterion optimization has no single global optimum, but a set of different ones. As no general method exists for multi-criteria problems, the usual solution method converts the set of metrics into a single aggregated metric. The first fundamental question is how to generate a common aggregated cost value from the set of available cost

DOI: 10.14750/ME.2013.009

components. Usually, the weighed sum of the components is used for the aggregation operation, i.e.

$$\overline{x} = \sum_{i=1}^{n} w_i \cdot x_i. \tag{5.2}$$

The main question of this approach is the appropriate selection of the weight factors as they determine the importance of different metric components. During the weight selection, we should care to normalize the different dimensions of the different metrics. Beside the weighted sum approach, the literature contains some other aggregation methods too (Ghosh and Nath 2004) such as

- ε-perturbation
- goal programming
- Tshybeshev-method
- min-max method.

In the optimal case, the result of the single-criterion optimization yields a Pareto-optimum value (Ghosh and Nath 2004). In this approach, a dominance relation is defined on the metric (cost) vector. A Pareto-optimal solution is such a point in the search space which is not dominated by any other points in the space. The dominance determines a partial ordering among the elements of the search space, thus the elements can be structured into a lattice.

Considering the applied optimization methods, the following main categories can be defined:

- **brute force method:** all possible solution candidates are enumerated and evaluated
- **direct formal optimization:** the cost function is a continuous, differentiable function; the derivate of the cost function is equal to zero at the optimum points
- **evolutionary methods:** the combination of stochastic and direct methods is used to localize the different local optimum in the hope to converge for global optimum
- **specific heuristic method:** the specialty of he investigated problem domain is incorporated into the optimization method.

Nowadays, the evolutionary methods are the most widely investigated methods in the literature (Ghosh, Dehuri, and & Ghosh 2008).

5.2 Optimization in POS tagging

The morphology information alone is usually not enough to infer the semantic role of the word within the text. The sentence level structure encodes the required additional information to determine the semantic role. At the sentence level, the grammar role determines the functional role of the word together with the relationships to the other words. A human sentence can be interpreted as a complex structure containing parts with different semantic grammatical roles. A usual sentence contains for example a predicate part and a subject component. These roles are the part of speech (POS) units of the sentence. A simple parser usually uses about 20 different roles while the more sophisticated models distinguish more than a 100 POS variants.

DOI: 10.14750/ME.2013.009

The set of common POS roles includes, among others, the following items: NN (singular noun), NNS (plural noun), VB (verb base form), VBD (verb pass tense), PN (personal pronoun), AT (article), IN (preposition), RB (adverb) or JJ (adjective) (Manning and Schütze 1999). In the tagging process, the words of an input sentence will be assigned to a corresponding POS unit.

The main difficulty in the tagging process arises from the ambiguity of the words: a word may have different meanings and a word sequence may have different POS tagging. The next sentences give examples for these difficulties:

Peter [NN] move [VB] into [IN] a[AT] new [JJ] city [NN]

(There is) [VB] a [AT] move [NN] in [IN] (New York) NN

5.2.1 Markov POS tagger

The usual approach in tagging is to consider the word sequence as a random Markov process. The Markov chain model assumes that the state value at time point *t* depends only on the state value of the previous time point *t-1*. The second key assumption is that this probability is stationary, i.e. it is the same for every *t* value. This means that the probability of a tag sequence can be given with the neighborhood probabilities (i.e. a state depends only on the parent state)

$$p(t_{1..n}|w_{1..n}) = \frac{p(w_{1..n}|t_{1..n})p(t_{1..n})}{p(w_{1..n})} = \frac{\prod_i p(w_i|t_i)p(t_i|t_{i-1})}{p(w_{1..n})}, \tag{5.3}$$

where the symbols are as follows

- $w_{1..n}$: word sequence between positions 1 and *n*
- $t_{1..n}$: tag sequence between positions 1 and *n*
- w_i : word at positions *i*
- t_i : tag at positions *i*.

Thus the most possible tag sequence belonging to a given input word sequence is calculated with

$$t_{1..n} = arg\max_t \prod_i p(w_i|t_i)p(t_i|t_{i-1}). \tag{5.4}$$

The probability values between the neighboring states are estimated with the relative frequency values. An efficient implementation of tagging optimization is the Viterbi algorithm (Freitag and McCallum 2000). The Viterbi algorithm uses a dynamic programming approach to find the optimal transition route. It stores in a matrix the cost value for every possible intermediate states. An intermediate state is characterized with a time index and with a tag index. Based on the stationary assumption, the $p(t_i|t_{i-1})$ value is independent from the position, it depends only on the tag values, i.e.

DOI: 10.14750/ME.2013.009

$$p(t_i|t_{i-1}) = p(t^k|t^l). \tag{5.5}$$

where t^k and t^l denote the corresponding tag values in the tag dictionary. The cost of the intermediate state with index *(i+1, j)* is calculated with

$$c(i+1,j) = \max\{c(i,l) \cdot p(t^j|t^l)| \cdot p(w_{i+1}|t^j)\}. \tag{5.6}$$

The HMM method belongs to the family of generative models, where the joint distribution is the base formula to calculate the conditional probabilities. Another approach is represented by the discriminative model which focuses directly on the conditional probability. The Linear-chain Conditional Random Field (LCRF) (Lafferty, McCallum, and Pereira 2001) is an efficient alternative to the HMM methods.

5.2.2 Linear-chain Conditional Random Field

The main idea of CRF is to segment the variables into smaller disjoint groups, where every group is independent of the other groups. This process is called factorization. In LCRF, the conditional probability is calculated with (Sutton and Mccallum 2012)

$$p(t_{1..n}|w_{1..n}) = \frac{1}{Z(w_{1..n})} \prod_i \exp\left\{\sum_k \theta_k f_k(t_i, t_{i-1}, w_i)\right\}, \tag{5.7}$$

where *Z()* is the normalization function, *f()* is the feature function and θ is the parameter vector. The domain of feature engineering refers to the selection of appropriate feature functions. The number of feature vectors may be very large in practical applications, for example in (Sha 2003) about 3.5 million features were used. For a given feature and vector set, the above mentioned Viterbi algorithm can be used to determine the most probable tag sequence.

In LCRF, a separate step is the selection of the appropriate parameter vector. For calculation of the maximum likelihood feature vector, a numerical optimization is used. The proposed method usually optimizes the conditional log likelihood value. To avoid overfitting, the following regularized log likelihood is suggested for parameter optimization

$$\varepsilon(\theta) = \sum_j \sum_i \sum_k \theta_k f_k\left(t_i^j, t_{i-1}^j, w_i^j\right) - \sum_j \log Z\left(w_{1..n}^j\right) - \sum_k \frac{\theta_k^2}{2v}, \tag{5.8}$$

where the upper index *j* denotes the training data index and ν denotes a penalty factor. At the optimum parameter value, the derivate of the objective function is equal to zero. The solution of the system of equations,

$$\sum_j \sum_i f_k\left(t_i^j, t_{i-1}^j, w_i^j\right) - \sum_j \sum_i \sum_{t,t'} f_k\left(t, t', w_i^j\right) p\left(t, t', w_i^j\right) - \frac{\theta}{v}, \tag{5.9}$$

yields the optimum parameter vector.

DOI: 10.14750/ME.2013.009

The main shortcoming of this model is that it considers a sentence as a linear chain. The classic grammar models like TAG (tree adjoining grammar) are based on this approach. The TAG formalism proposed by (Joshi, Levy, and Takahashi 1975) defines initial and auxiliary trees. A tree encodes the set of possible sequences, where an element of the sequence can be replaced with other subsequences (adjunction operation). For languages with strict word order, the sequence model is a good approach. However, the analysis of the sentence structures in Hungarian language shows that the sequence orientation is not the perfect model for languages with no dominant word orders. In these languages, all permutations of the words may be valid. The explicit encoding of all possible permutations would result in intractable grammar trees. To provide a more suitable formalism, a graph oriented model is proposed in our system.

Another key problem in the tagging operation is finding the correct segment boundaries. A word belonging to different segments may have the same grammatical properties. In the sentence

Tegnap telefonált Peti Zoli barátjának.
(Peter phoned Zoli's friend yesterday.)

After the morpheme analysis, both words Peti and Zoli will be labeled as NN, thus both are candidate subjective of the sentence. The correct segmentation should find the followings segments within the sentence

- predicate : telefonált
- subject : Peti
- dat_object : Zoli barátjának
- adverb: tegnap.

Segments may be of simplex structure or of complex one. For example, in a higher order statement a sub-statement can be used as a POS unit

Mondtam neked, hogy gyere pontosan.
(I have told you to come in time.)

This examples shows also the fact that sometimes some POS roles are implicitly given, they have no explicit word representative. The subject I is encoded in the inflection of the verb. Thus the POS segment structure of the sentence is:

- predicate : mondtam (past)
- subject : én (hidden , implicit)
- dat_object : neked
- object : hogy gyere pontosan
- object-predicate: gyere
- object subject: te (hidden, implicit)
- object-adverb: gyorsan.

DOI: 10.14750/ME.2013.009

The management of free word-order was addressed in some previous grammar models too. One important group of these kinds of models is the family of dependency grammars. The dependency grammar (DG) (Tesniere 1959) is based on modeling the dependency between different POS units. For example, a predicate unit requires a subject and an object POS unit. The dependency is an asymmetric relationship, the head corresponds to the independent part of the relationship. A similar approach is presented in the word grammar (Hudson 2007), which is based on the dependency grammar. In this model, the language can be represented with a network of propositions. This approach can be applied, among others, in the neurolinguistics domain.

5.2.3 Proposed tagging method

The semantic structure of sentences is given with a multi-level graph. In the graph model a graph is given with

$$G = \{l, K, G_N, G_D, G_P\}, \tag{5.10}$$

where G_N denotes the set of nodes. A node may be either a simple node or a graph again. There is a special node called kernel node K. The symbol l is for the identifier label of the graph. Every graph corresponds to a segment of the sentence. Within the segment, there is a dependency relationship between the kernel node and the other nodes. The kernel node is always the source node of the dependency relationship. The G_D set of edges corresponds to the dependency relationship between the items,

$$G_D = \{K \to L | L \in G_N\}. \tag{5.11}$$

In the graph, every non-kernel node depends only on the kernel. Each node in the graph is given with a set of attribute tuples. An attribute tuple includes the following elements:

- a role name of the node
- morpheme tags of the corresponding words (A_m)
- probability of a given morpheme tag set (A_p)
- flag to denote whether the node, role is optional or not (A_o).

The set G_P describes the precedence relationship between the elements of $\{G_N \cup K\}$. The set G_P may be empty denoting that no word order rule can be discovered, every word order is valid. To each element of G_p, a probability value is assigned.

For the sample sentence, the morpheme analyzer returns the following morpheme structure:

- tegnap : [FN] + [NOM] | [HA]
- telefonált : [IGE] + [MIB] + [e3]
- Zoli : [FN] + [NOM]
- Peti : [FN] + [NOM]
- barátjának : [FN] + [PSe3] + [DAT].

DOI: 10.14750/ME.2013.009

Based on this list, a graph model can be generated as it is shown in Figure 5.1. The solid line corresponds to the dependency relationship, while the dashed line shows the precedence relationship among the graph nodes.

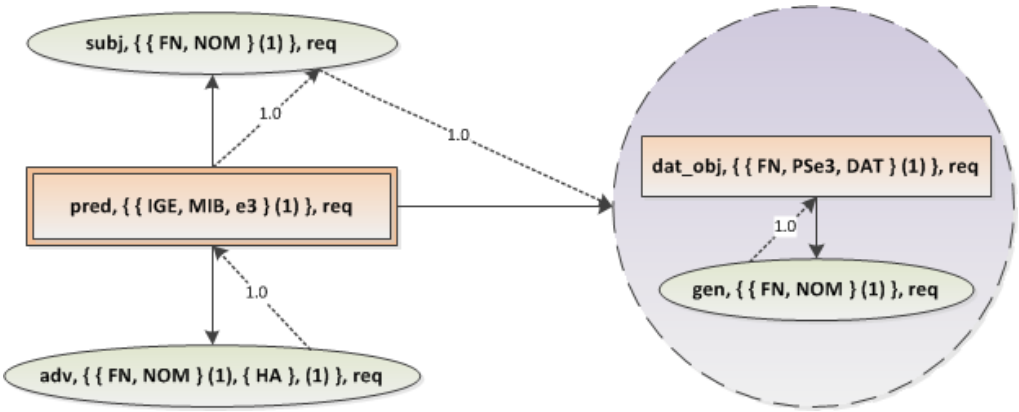

Figure 5.1. *Sample sentence graph for sample sentence*.

The graph structure contains the segmentation structure of the sentence. The kernel node of the main segment is a predicate node (double-line border) and the kernel of the sub-segment is a non-predicate word (single-line border).

In the graph model, the words assigned to the graph nodes correspond to the elements w_i of the sentence. The role name of the edges pointing to the node denotes the tag name t_i. The morpheme structure is a set of atomic morpheme units. The conditional probability $p(a|r)$ for every morpheme unit can be calculated with the sum of the corresponding tuple probabilities. Based on the graph, the tag role value can be estimated both from the morpheme structure and from the precedence relationship. For the morpheme-based probability, we use the Bayes-formula

$$p_m(r|w) = \left\{\frac{p(r)\cdot\prod_{a\in w} p(a|r)}{p(w)}\right\}. \tag{5.12}$$

Having the independency assumption, the morpheme probability for a tag sequence is equal to the product of the probabilities of the components as

$$p_m(r_{1..n}|w_{1..n}) = \prod_i p_m(r_i|w_i). \tag{5.13}$$

The precedence probability for the tag sequence is given with

$$p_p(r_{1..n}|w_{1..n}) = \prod_{i,j} p_p(r_i r_j), \tag{5.14}$$

where p_p comes directly from the graph model. The sum of the morpheme and of the precedence probabilities defines the weight of a tag sequence by

$$p(r_{1..n}|w_{1..n}) = p_m(r_{1..n}|w_{1..n}) + p_p(r_{1..n}|w_{1..n}). \tag{5.15}$$

DOI: 10.14750/ME.2013.009

For a given word sequence, the algorithm selects the tag sequence of maximum weight by

$$r'_{1..n} = \arg\max_{r}\{p(r_{1..n}|w_{1..n})\}. \tag{5.16}$$

In order to reduce the cost of the brute force search operation, a dynamic optimization method can be used similar to the Viterbi algorithm.

5.3 Graph matching algorithms

In grammar analysis, the tree structure is the most important representation structure. In our framework, trees are used to store the discovered part of speech units and the dependency relationships between the units. The schemata of different functions are also stored in tree structure. The function space is described with a set of corresponding schema trees. During the mapping of the POS-tree into the function-trees, a tree matching algorithm is applied. The goal of this phase is to find the best matching schema tree for the input POS tree.

Regarding the matching algorithms, we can distinguish

- exact methods
- nearest neighbor (fuzzy) methods.

In the case of exact search, the target tree must be the same as the query tree. In our problem domain, this is not the case, as the schema tree contains all possible elements from which some of them are optional. Instead of managing all possible combinations of required and optional elements and performing exact search, our implementation stores an extended schema and uses a nearest neighbor search (NNS).

On the field of NNS, the methods depend on the object domain. If the objects are members of a one-dimensional vector space, the usual index structures, like B-tree indexing (Kovács 2004) can be used. In the case of multi-dimensional vector space a more complex structure is required. The R-tree or S-tree structures (Guttman 1984) are common solutions for this problem. In the R-tree structure, the object space is divided into a hierarchy of rectangular areas. The nodes of the R-tree corresponds to the rectangles, a child node corresponds to a sub-rectangle. For the administration of the child nodes, the coordinate values of the corners are stored, thus for a query object, the container child can be selected with a simple computation.

In the case of general metric spaces, where the objects cannot be represented with appropriate vectors, distances between the objects are the only information we can work with. The VP-tree structure (Kovács 2010) or one of its variants is a widely used structure for this case. In VP-tree, during the segmentation of the object space, the distance value to a selected (vantage) point is used to separate the objects into two subsets.

Beside the VP-tree index structure, another option is the k-d-tree approach. The k-d-tree has the same structure as the one dimensional index tree, but here different levels of the tree are assigned to different dimensions. The assignment is performed in cyclic manner.

DOI: 10.14750/ME.2013.009

A special variant of the k-d-tree is the prefix tree (Weiner 1973). In the prefix tree, different levels correspond to different positions within the structure. Every position within the complex structure can be used as a separate dimension.

As it can be seen, in each tree index structure, the distance values between the objects are the key elements for the positioning of objects. The distance between two tree objects can be calculated with different methods. Usually, a metric distance is preferred, where

$$\begin{aligned} d(x,y) &\geq 0, \\ d(x,x) &= 0, \\ d(x,y) + d(y,z) &\geq d(x,z). \end{aligned} \tag{5.17}$$

For complex structures, it is usually hard to find an appropriate metric distance. The distance between two structures is based on the aggregation of the distances between the components. Taking sets, the usual distance is the Hausdorff distance (Rucklidge 1996) which is calculated with

$$d(X,Y) = max\{sup_x inf_y d(x,y), sup_y inf_x d(x,y)\}. \tag{5.18}$$

From our viewpoint, the drawback of the Hausdorff distance is that the distance value depends only on some selected elements, the similarity between the majority of the elements is not important.

Another approach is the mapping-based distance measure. Having a structure preserving mapping between the two tree structures, the number of paired nodes determines the distance value between two elements

$$d(X,Y) = 1 - \frac{|\{x|\exists y\colon (x,y) \in P\}| + |\{y|\exists x\colon (x,y) \in P\}|}{|X| + |Y|}. \tag{5.19}$$

In this interpretation, the P symbol denotes the set of matching pairs. For every pair *(x,y) x* is a node in tree *X* and *y* is a node in tree *Y*. A pair *(x,y)* is a matching pair, if

- the labels of both nodes are the same ($l(x) = l(y)$), and
- the parents are members of the same pair: $\big(p(x), p(y)\big) \in P$.

In the proposed system, the prefix tree approach was implemented. The root node of the index tree is an abstract node. The root node in the prefix tree has as many child nodes as many different labels occur in the root elements of the POS-trees. The child nodes of any parent node in the prefix tree correspond to the different possible label values in the child instances of the parent instances. A child node is tested only if its parent has been matched already. This algorithm enables a reduced fuzzy matching as

- the root must be matched always
- if a node is matched, its ancestors are matched, too.

DOI: 10.14750/ME.2013.009

For efficiency analysis, we calculate the cost function of the proposed method. For quantitative analysis, we use the following notations:

- M : size of the query tree
- N : number of schema trees in the database
- P : number of different predicate labels
- L : average number of child nodes
- K : number of different child labels for a given parent label.

In testing a query tree, the schema trees can be filtered by the predicate label. The subset of schema trees related to a given predicate label can be selected with a

$$\mathcal{O}(\log(P)) \tag{5.20}$$

execution cost. The child nodes are tested with a cost

$$\mathcal{O}(L \cdot \log(K)) \tag{5.21}$$

taking two levels of child nodes, the total time cost is estimated with

$$\mathcal{O}\left(c_1\left(\log(\mathrm{P}) + \frac{N}{P}(L \cdot \log(K) + K \cdot \log(\mathrm{K}))\right)\right), \tag{5.22}$$

where c_1 denotes a node matching cost. The storage cost can be estimated with

$$\mathcal{O}(P \cdot L \cdot K). \tag{5.23}$$

For the VP-tree model, the search operation has a

$$\mathcal{O}\left(c_1 \cdot \left(R \cdot \log(N) \cdot (L \cdot \log(K) + K \cdot \log(K))\right)\right) \tag{5.24}$$

cost. In the formula, R is the replication factor. As it is not guaranteed that a top-down search path yields the nearest element, this path may be repeated several times (Kovács 2010). If P is relatively large compared with N, the prefix-tree structure provides a more efficient execution cost.

5.4 Optimization tasks in function mapping

The difficulty of function mapping is caused by ambiguous mapping between sentence analysis and function signatures *(FS)*. There are concepts in the domain which can fit into many *FS*, predicates do not assign an individual *FS*, there are missing parameters in the sentence and similar troubles make mapping much harder.

To be able to manage these situations the mapping process should be formalized. A mapping between the sentence analysis tree and *FS* can be defined by

$$S \to f_i \mid F = \{f_i\} \tag{5.25}$$

DOI: 10.14750/ME.2013.009

where S is the sentence analysis tree and f_i is the i^{th} function signature from set of function signatures F. Let us define a fitness function Φ which calculates the correspondence value between sentence S and a given function signature f_i as

$$\phi_i: (S, f_i) \to \mathbb{R}. \tag{5.26}$$

The goal of function mapping is to find the winner function Φ_w which has the highest fitness value

$$w = \arg\max_{i} \phi_i \,, i = 1 \ldots |F|. \tag{5.27}$$

Constructing ϕ_w the proper fitness function is the key component of this module, where the following statements have to be taken into consideration:

1. A sentence analysis tree has only one predicate and can have more other concepts. (We work only with simple, extended sentences.)
2. A predicate can be assigned to one or more *FS*.
3. A concept of the sentence analysis tree should be mapped to one of the parameters of *FS*.
4. A concept can be mapped only to one parameter of *FS* in the same time.
5. A concept may not belong to any parameter.
6. A *FS* can have required and optional parameters, where finding of required parameters are more important. If a required parameter is missing, the method belonging to FS cannot be executed.
7. If a parameter has dependency node, only sentence analysis node with proper parent can be mapped.

Let us define a sentence analysis tree as

$$S = (C, \to), C = \{P\} \cup \{c_i\}, i = 0 \ldots n, \to \subseteq C \times C, \tag{5.28}$$

where P is the predicate concept and c_i is a non-predicate concept, $\to$ is the link between concepts in the tree. The following statements are valid for analysis tree.

$$\forall c_i \in C \Longrightarrow P \to c_i \text{ and } \forall c_i, c_j \in C : c_i \to c_j \Rightarrow c_j \nrightarrow c_i. \tag{5.29}$$

The set of *FS* is defined by

$$F = \{f_i\}, f_i = (\Gamma, \Pi), \Gamma = \{\gamma_i\}, i = 1, \ldots, n, \Pi = \{\pi_j\}, j = 1, \ldots, m, \tag{5.30}$$

where Γ is the set of predicates and Π is the set of parameters.

A parameter defined in (5.31) has some attributes like name, position, type and mandatory. In construction of fitness function only the mandatory property can have significant role the others are used in preparing execution of method belonging to given *FS*.

$$\pi_i = (m, \Theta), m \in \{0,1\}, \Theta = \{\theta_i\}, i = 1, \ldots, n, \tag{5.31}$$

DOI: 10.14750/ME.2013.009

where Θ means the constituents of parameter. A constituent may also have zero or more dependencies

$$\theta = \{\delta_i\}, i = 0 \dots n. \tag{5.32}$$

Fitness value points can be given for a *FS* in the following scenarios:

1. Predicate of the sentence is found among predicates of given *FS*.
2. Non predicate concept can be mapped to any of the parameters of the given *FS*.
3. If a parameter has a dependency, fitness score can be applied only if the concept and its parent together are fit.
4. Higher score will be applied if the actual parameter is required.

Since a predicate of S can be matched for many functions, the relevance is an important factor when choosing the proper function. The less number of functions matches with a given predicate the higher is the predicate relevance. A Tf-idf algorithm can be used to determine the relevance of a predicate by

$$\begin{gathered} tf(P, f_i) = freq(P, f_i), \\ idf(P, F) = \log \frac{|F|}{|\{f_i \in F : P \in f_i\}|}, \\ tfidf(P, f_i, F) = tf(P, f_i) \times idf(P, F). \end{gathered} \tag{5.33}$$

In mapping score system we have 3 different scores: predicate relevance *(x)*, number of found required parameters *(y)*, number of found parameters *(z)*.

The resultant value can be computed with weighted values of the above scores as

$$\Phi_i = \alpha x_i + \beta y_i + \delta z_i. \tag{5.34}$$

5.5 Summary of results

In this chapter the key processes of NLC was examined concerning cost and accuracy. The POS tagging, sentence analysis and function mapping has been analyzed since these tasks give the highest part of the cost. The accurate solution to tasks is also relevant, since e.g. if POS tagging produce false result, the sentence analysis and function mapping will also fail. The optimization results can be summarized as follows.

Thesis 3.

I have determined the key cost components in the natural language controlling framework. The proposed multi-criteria cost model includes execution time, accuracy and resource consumption factors. Algorithm optimization was performed for the POS tagging, sentence analysis and function mapping modules. The proposed graph-based algorithm provides an efficient execution for the implementation of the framework in large-scale domains.

DOI: 10.14750/ME.2013.009

6 Applications of Theoretical Results

The last task of my research was to implement the natural language framework modules and to present two sample applications that use the proposed framework in order to verify the applicability of theoretical results. The sample applications have different domains: robot controlling and navigation domain and they have their own function sets. The operability of applications proves the domain-adaptivity property of controlling framework. Applications receive Hungarian text input the source of which can be either a keyboard or a speech recognizer.

6.1 Natural Language Controlling framework

The process of building applications using natural language input is composed of three stages. First, modules of the framework should fully or partly be adapted. Adaptation means implementing language-dependent modules for the corresponding language or adding, such as text processing tasks depending on the nature of the application, e.g. city, address recognition in case of navigation-related texts.

The framework is planned to be able to use existing NLP engines for making text segmentation, morphology analysis, named entity recognition, etc. Modules of the framework provide well-defined interfaces which can individually be implemented and used in concrete applications. All modules have a built-in implementation for Hungarian language which can arbitrarily be replaced by other implementations.

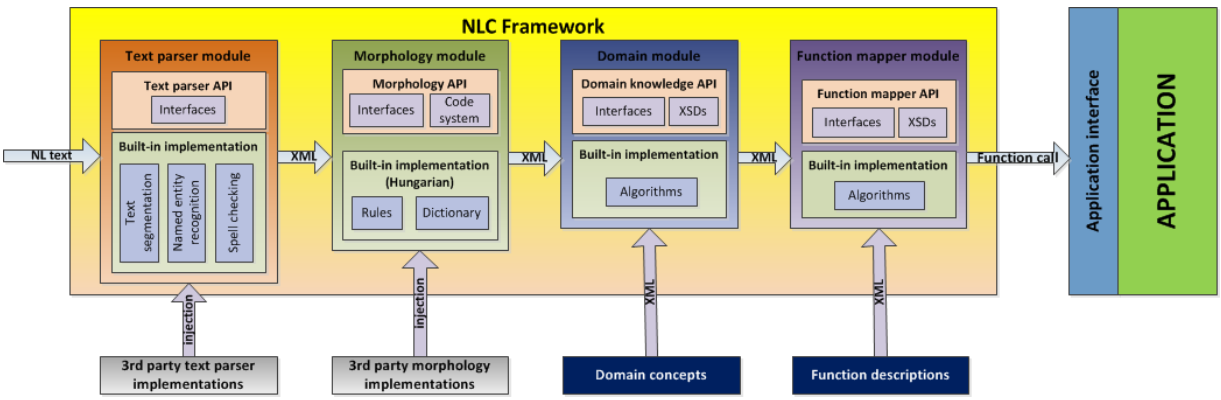

Figure 6.1. *Operational model of framework*

The implementation of the first two modules of the framework, the text parser and morphology modules, should be redeveloped in order to be able to be applied for different languages not only for the built-in Hungarian. The implementation of the last two modules which process semantics and application-function mapping can universally be used, since it has the same tasks for all languages. The only task in these modules is the adaptation of the

DOI: 10.14750/ME.2013.009

domain knowledge and the function descriptions which are interpreted dynamically. Therefore the components of the framework do not have to be changed, only the proper configuration has to be set.

The framework is developed in Eclipse Indigo SR2 IDE using Java version 1.6.0_45. The operational model of the framework is shown in Figure 6.1. The modules can be accessed from the following maven repository which is placed on one of department's servers: https://193.6.5.42/nexus/content/groups/nli.

The following table summarizes the needed maven information which can be set as dependencies in developing natural language applications using the framework.

Table 6.1. *Maven information of NLC framework modules*

Module/jar	Maven information

Text parser - API	`<dependency>`
	`<groupId>hu.miskolc.uni.iit.nli</groupId>`
	`<artifactId>nli-text-api</artifactId>`
	`<version>1.0.0-SNAPSHOT</version>`
	`</dependency>`

Built-in text parser implementation	`<dependency>`
	`<groupId>hu.miskolc.uni.iit.nli</groupId>`
	`<artifactId>nli-text-service</artifactId>`
	`<version>1.0.0-SNAPSHOT</version>`
	`</dependency>`

Morphology - API	`<dependency>`
	`<groupId>hu.miskolc.uni.iit.nli</groupId>`
	`<artifactId>nli-morpho-api</artifactId>`
	`<version>1.0.0-SNAPSHOT</version>`
	`</dependency>`

Built-in morphology implementation	`<dependency>`
	`<groupId>hu.miskolc.uni.iit.nli</groupId>`
	`<artifactId>nli-morpho-service</artifactId>`
	`<version>1.0.0-SNAPSHOT</version>`
	`</dependency>`

Domain knowledge - API	`<dependency>`
	`<groupId>hu.miskolc.uni.iit.nli</groupId>`
	`<artifactId>nli-domain-api</artifactId>`
	`<version>1.0.0-SNAPSHOT</version>`
	`</dependency>`

Domain knowledge implementation	`<dependency>`
	`<groupId>hu.miskolc.uni.iit.nli</groupId>`
	`<artifactId>nli-domain-service</artifactId>`
	`<version>1.0.0-SNAPSHOT</version>`
	`</dependency>`

Function mapper - API	`<dependency>`
	`<groupId>hu.miskolc.uni.iit.nli</groupId>`
	`<artifactId>nli-func-api</artifactId>`
	`<version>1.0.0-SNAPSHOT</version>`
	`</dependency>`

Function mapper implementation	`<dependency>`
	`<groupId>hu.miskolc.uni.iit.nli</groupId>`
	`<artifactId>nli-func-service</artifactId>`

DOI: 10.14750/ME.2013.009

	<version>1.0.0-SNAPSHOT</version> </dependency>

6.1.1 Text parser module

The text parser module has the following subtasks which should be implemented:

- **Sentence/word segmentation, parsing:** the incoming text should be parsed into a sequence of sentences and the sentences should be divided into sequences of words
- **Named entity recognition:** special words like cities, companies, numbers, dates, etc. should be labeled with a category tag
- **Spell checking:** the mistyping of individual words should be analyzed by a spell checker.

Since each language has its own vocabulary, boundaries, named entities, etc. this module needs to be implemented fully for each. The framework should provide well-defined interfaces for the above tasks and the proper implementation should be injected when the application is built. Since the framework uses Spring 3 framework, dependency injection can be achieved easily with some XML configuration.

Text parser module provides the following interfaces:

Figure 6.2. *Interfaces of text parser module*

The module uses the actual implementation of the previously defined interfaces and produces the XML output from the natural language text input in following steps

DOI: 10.14750/ME.2013.009

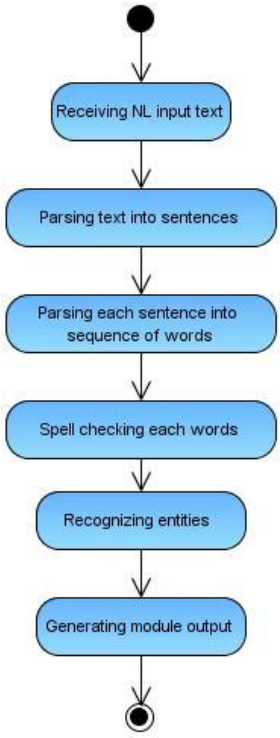

Figure 6.3. *Steps of text parsing*

The output of the module contains the detected sentences, each corrected word of sentences in separate nodes and entity labels for words if they exists. XSD 0.1 in Appendix B is used for the validation of the output of the text parser module.

6.1.2 Morphology module

Morphology module gets the output of the text parser module as an input. The interface of the current module contains the following functions:

- **Word analysis:** words of the incoming sentence should be stemmed, resulting the stem and suffixes of word
- **Word inflection:** stems can be inflected using given rules, e.g. for answer generation.

Morphology analysis is a complex task which depends on the grammar rules of the given language. A completely general solution cannot be developed for all languages in one, since the syntax and grammar is generally differ between languages. Therefore, morphology module should be implemented for each input language and the framework uses the injected implementation of the interface in Figure 6.4.

The output of morphology module should contain all possible analysis for each word, since in morphology level it cannot be decided which one is the correct alternative; it is one of the tasks of semantic level.

DOI: 10.14750/ME.2013.009

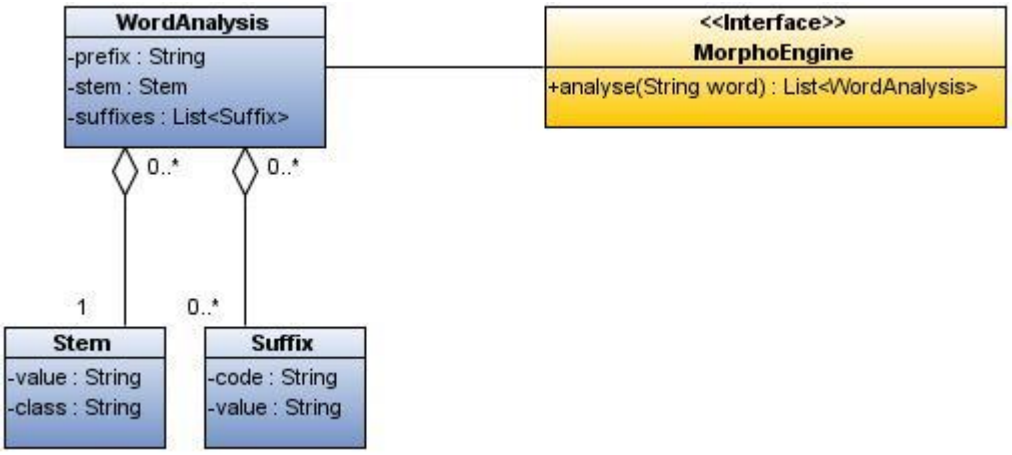

Figure 6.4. *MorphoEngine interface and belonging models*

Regarding language dependency, the morphology module manages two kinds of code tables:

- **Word codes:** each stem concatenated with prefix has a unique code and the actual value in the given language is stored besides them
- **POS codes:** part of speech tagging should use codes form this set which consists of base codes and suffix codes also. List of POS labels can be seen in Appendix A.

The development of the morphology module should take care for the following subtasks:

- Adding word values to existing word codes. If a word code would be missed, the new record should be added to the list with null values for other languages. It can be changed later.
- Implementing *MorphoEngine* interface with existing NLP engines/modules or with a self-developed one.
- Configuring the application to use the proper implementation.

Our built-in morphology engine uses a Hungarian dictionary from *Szószablya* project (Halácsy et al. 2003) and the analysis is based on database query operation. It consists of not only the words of the domain but also of the most of Hungarian words.

The output of module is also in XML form, which can be validated with XSD 0.2 in Appendix B.

6.1.3 Adaptation of domain knowledge

The next step in natural language text processing is the sentence analysis which utilizes the morphologically labeled sequence of words. Unfortunately, only this information is not enough for the proper and accurate analysis, because there are many situations, where e.g. the suffix does not assign the role in the sentence. Even people can only make accurate sentence analysis when they know the concepts and their relations to each other.

DOI: 10.14750/ME.2013.009

The third module of the framework is prepared to make the sentence analysis of the incoming sentence only if the concepts of the domain and their relations are well-known. The format of description of concepts and relations is also XML and should contain data as

- Abstract, predicate, attribute and individual concept descriptions
- Relationships between previously defined concepts.

The details of descriptions can be read in Chapter 4, the only explanation is that the code values, which belong to stems, should be used in application descriptions instead of concrete stem values of specified language. The framework provides a method which can read the XML description, parses the tree and makes the suitable model in memory to be able to make the sentence analysis. Application developers should use the following classes for this purpose:

Figure 6.5. *Knowledge loader class of Domain module*

Multiple knowledge files can be loaded together if the domain concepts are split into more files. At the same time, the whole knowledge base can be cleared by calling the *clear()* method.

Another knowledge type should be loaded into the system which is the constituent list of the language. It contains information defined in Chapter 4, the only extension is that here the technical codes should also be used in language-dependent parts, such as values of interrogatives to ensure the language independency of the module.

The output of the domain module consists of the sentence analysis tree of the incoming sentence. The nodes of tree are the found concepts with additional information like

- in-sentence role
- concept value
- usage in later processing.

The relation between concepts is derived from the structure of the analysis tree. The root node of analysis is usually the predicate concept and the belonging concepts are placed in its sub-tree. Each parent-child relationship is represented in a node-child node structure in the analysis tree. One of the most frequently applied parent-child patterns is the attributive one. The schema XSD 0.3 in Appendix B describes the valid structure of the sentence analysis.

6.1.4 Adaptation of function descriptions

The last task in tuning the NLC framework for a specific application is the construction of application function descriptions (AFD) and loading them into the framework. AFDs denote the context of real functions as described in Chapter 4. Arbitrary AFDs can be loaded into the

DOI: 10.14750/ME.2013.009

framework using the class in Figure 6.6 or can be unloaded one at a time or altogether. Certainly, all AFD files are validated with the corresponding XSD in the loading phase.

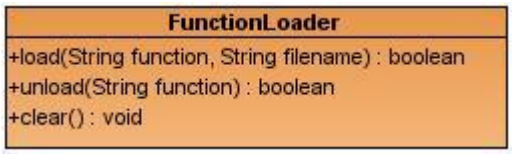

Figure 6.6. *Function loader class of function mapper*

For extending applications with natural language control an interface application should usually be developed classes and methods of which are referred in AFDs since most of the applications have no public APIs that can be called directly from the function mapper module.

In the next sections two sample applications will be introduced which use our NLC framework adapted to different domains.

6.2 Robot controlling application

Communication with most of computer systems happens via user-friendly but not natural ways. People have to fill out forms, click buttons, type instructions through a keyboard instead of saying commands in a natural language to the system. Natural language processing is a more and more popular area of artificial intelligence and it has a wide application area like robot controlling. The combination of these two areas improves significantly the functionality of robotic assistants which became more common in environments such as offices, houses and industries. Increasingly, the communication with robots wanted to be much easier and in more natural ways like via spoken dialogs.

There are numerous researches which combine the fields of natural language processing and robotics. Sondheimer (Sondheimer 1976) is focused already in 1976 on the problem of spatial reference in natural language machine control. Nilsson's SHAKEY system (Nilsson 1984) from 1984 is able to understand simple commands given in a natural language. Sato and Hirai (Sato and Hirai 1987) worked on language-aided instruction for teleoperational control, where specific words can be utilized to make the specification of teleoperational functions for instruction of remote robot system simpler. Torrance (Torrance 1994) made a natural language interface for an indoor office-based mobile robot in navigation area. Lobin (Lobin 1992) points out some theoretical aspects of natural language communication with robots from the perspective of computer linguistics. Badler (Badler et al. 1991), Chapman (Chapman 1991), Vere and Bickmore (Vere and Bickmore 1990) control autonomous agents within simulated 2D and 3D environments in a natural language.

RHINO (Burgard et al. 1998) is a robotic guide in a museum which can move and describe particular exhibits. It can only recognize simple phrases without supporting true dialogues. RHINO's ancestor was Polly, a vision-based robot which interaction mechanisms were more primitive. MAIA robot (Antoniol et al. 1994) could carry objects from one place to another. It could be controlled by simple spoken commands. Jiro-2 (Asoh et al. 1999) is a mobile office

DOI: 10.14750/ME.2013.009

assistant which can convey information and guide people through an office environment. A frame-based SDS is used by Jiro-2 and it communicates in Japanese. AESOP 3000 (Versweyveld March 1998) is a surgical robot which is controlled by voice in a heart surgery and does not provide a full SDS.

Robots in (Lemon et al. June 2001) and (Perzanowski et al. June 2001) use multi-modal interfaces which comprise both speech, keyboard and point-and-click input also. Accuracy of robot controlling systems can be increased using multi-modal interfaces contrary to unimodal systems. Flippo discusses multimodal interfaces detailed in (Flippo August 2003).

There are many robot controlling systems that work similar to our proposed system. Our frame-based dialog system differs mainly in the description of slots and mapping rules. We use sentence analysis-based mapping, where words are mapped to a slot not only by the word itself but considering in-sentence roles also. Concepts of the domain should be defined to achieve proper sentence analysis which cannot be made only with statistical methods.

The structure of application can be seen in Figure 6.7.

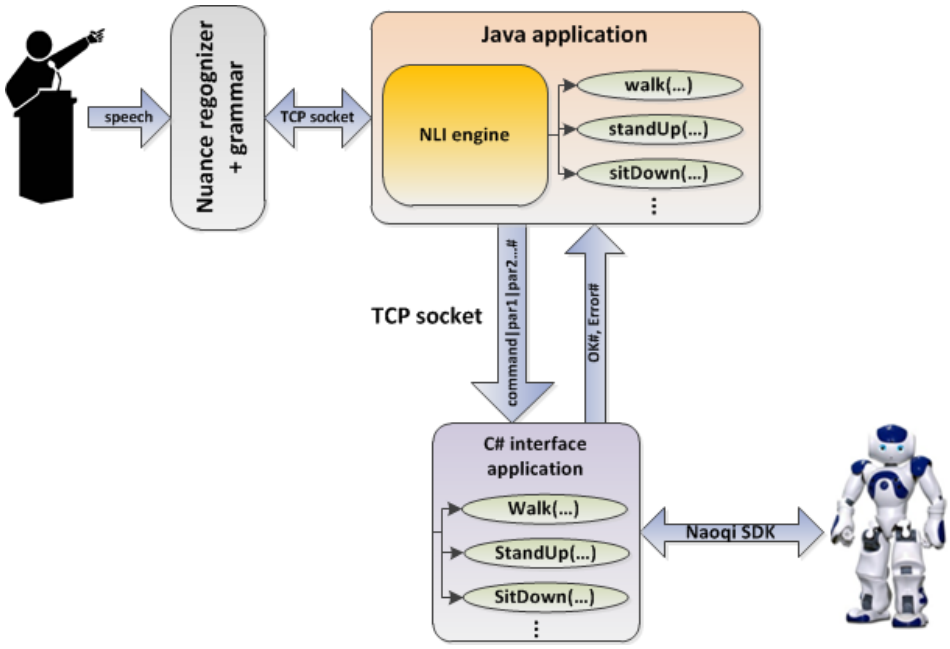

Figure 6.7. *Structure of Robot Controlling Application*

The highest user experience can be achieved, when the user can talk to the robot instead of writing and the robot executes the commands of the human. In case of Hungarian language the speech recognition, mostly the speaker-independent ASR, is in its childhood and it does not exist an exact, fast and accurate enough solution which can perform the properties of a real-time usage. There are some by-pass solutions using of which speech can be recognized with restrictions. Nuance (Nuance 2013) has some promising products like Nuance Recognizer or Dragon which supports Hungarian language increasingly.

DOI: 10.14750/ME.2013.009

In sample application the Nuance Recognizer is used to convert speech into written text. Well-defined grammar XML files should be built which describes the word sequences that the robot is able to recognize. The drawback of this solution is that only those sentences can be recognized which are defined in the grammar. The benefit of this representation is that the grammar can be arbitrarily recursive so the number of combinations can be increased. It has certainly price: the process of recognition will be slower.

The speech recognizer or the keyboard generates the input of the NLC framework which is well-adapted with domain knowledge and function descriptions.

6.2.1 Robot domain knowledge

The robot domain knowledge is quite limited, only those concepts and relations are added that are required to execute the function defined in function set. The following table summarizes the concepts of the robot.

Table 6.2. *Concepts in Robot Contoller Application*

Concept type	Values
Predicate concepts:	stand, sit, lay, sorry, wave, nod, exclaim, make excercises, see, close eyes, say hello, walk, turn, introduce, welcome, raise, lower, twist, look,
Abstract concepts:	body_part, limb, head, leg, arm, hand, elbow, direction, unit, person, side
Individual concepts:	forward, backward, left, right, back, up, down, here, around, meter, centimeter, degree, step, Ági, Peti, Szabi, Attila, left one, right one

Between abstract and individual concepts mostly *'isa'* connection is defined. For the direction abstract concept Figure 6.8 shows the individuals.

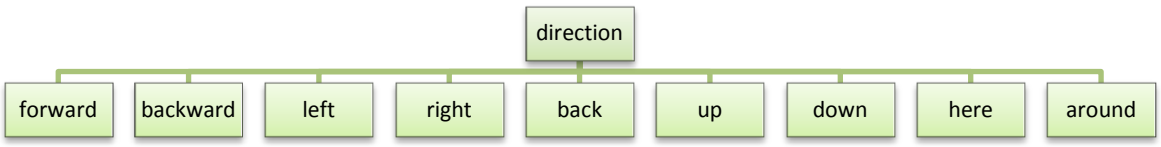

Figure 6.8. *Direction concept hierarchy*

The next example in Figure 6.9 shows the definition of 'walk' concept with belonging concepts. The sentence analysis can be fulfilled using such kinds of knowledge definitions.

DOI: 10.14750/ME.2013.009

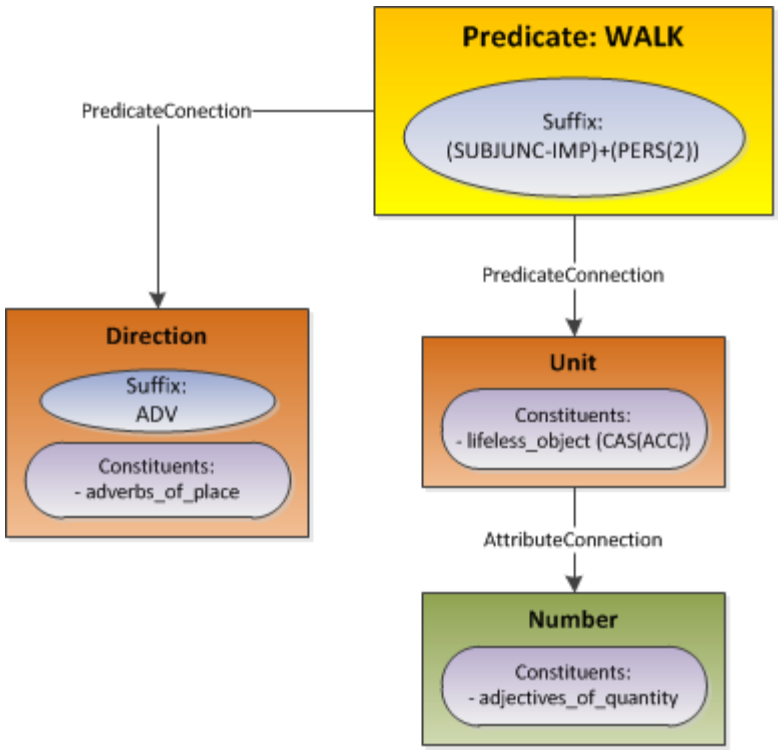

Figure 6.9. *Walk predicate description*

6.2.2 Function set

The domain knowledge is determined based on the set of available functions. If this set will be extended, the domain knowledge may be completed as well. In the sample application the following functions are realized to control the humanoid robot.

Table 6.3. *Functions of Robot Controlling Application*

Function	Description
Bow	Robot can bow its bust after introduction is completed.
HideVideo	Robot can „close its eyes” which means hiding camera window in application.
Introduce	Robot can introcuce itself with saying like: „Hello, my name is Nao.”
LayDown	Robot can lay down from any position.
Look	Robot can look in any direction with rotating its head.
Lower	Robot can lower its arms or legs in specified directions with specified scale.
No	Robot can move its head horizontally like showing “No”.
Raise	Robot can raise its arms or legs in specified directions with specified scale.
ShowVideo	Robot can “open its eyes” which means showing camera window in application.

DOI: 10.14750/ME.2013.009

SitDown	Robot can sit down from any position.
StandUp	Robot can stand up from any direction.
TaiChi	Robot can make some cool exercises such as TaiChi movements.
Turn	Robot can turn in any direction with specified angle.
Walk	Robot can walk in any direction until specified length.
Wave	Robot can wave its arm.
Welcome	Robot can welcome people to say: "Welcome XYZ."
Yes	Robot can move its head vertically to show understanding.

6.2.3 Communicating with Nao robot

Aldebaran's Nao (Aldebaran 2013) humanoid robot, shown in Figure 6.10, has a software development kit, which provides an application programming interface in several languages like C++, Python, C#. In the current version, Java API is also supported, but when application was implemented, there was no Java support yet. A by-pass solution had to be found to communicate with Nao from the natural language interface framework written in Java.

Figure 6.10. *Aldebaran's Nao humanoid robot*

The solution was to develop an interface application in C# which connects to the robot and calls its API methods. The C# application gets commands via socket communication where the interface application acts as socket server and the NLC framework has a socket client part.

The socket messages have a simple structure which consists of the command name and parameters. A sample socket message for the walk function is the following:

```
Walk|0.4|forward#
```

This instructs the robot to walk 40 centimeters forward. The delimiter is '|' between function names and parameters and sign *'#'* is the terminate symbol of the message. The interface

DOI: 10.14750/ME.2013.009

application calls the corresponding NaoQi API function and Nao makes the movement. In Figure 6.11 the user interface of Nao interface application is illustrated.

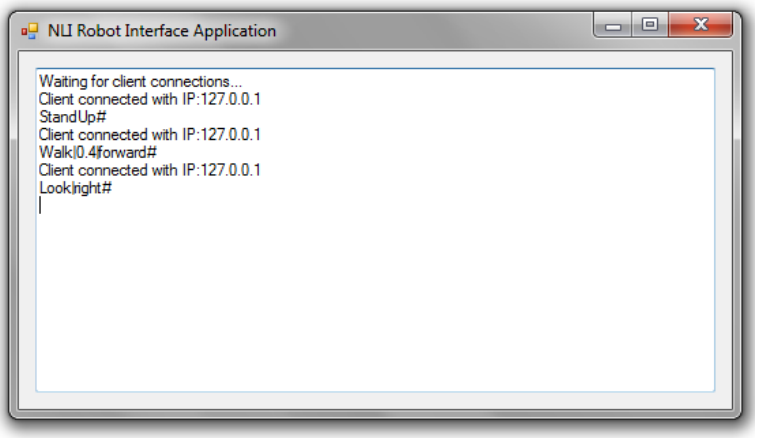

Figure 6.11. *Nao interface application*

6.2.4 User interface of Robot controlling application

Robot controller application has a simple graphical user interface described in Figure 6.12. It has two main parts: a dialog panel and an input area.

The conversation panel contains the questions, commands and the application answers in natural language format. A conversation entry consists of four parts:

- The entry type icon, which shows that the entry is textual or speech.
- The name of the user who has sent the conversation entry. It can be: Ember or Robot.
- The content of conversation entry in natural language format.
- The time of entry when it has been displayed in the panel.

Users can write natural language texts into the textbox of the input area. The command can be sent by pressing the *'Enter'* key or pressing the *'Küld'* button. After sending a command, it will be processed and answered by the application.

If the command has missing parameters or the application does not understand the command, a question entry will be shown in conversation panel.

DOI: 10.14750/ME.2013.009

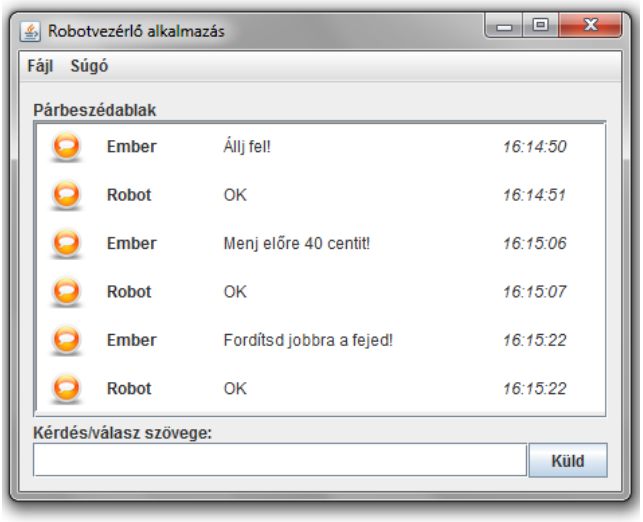

Figure 6.12. *Graphical user interface of robot controller application*

6.3 Navigation application using Google Maps

Controlling navigation systems with a natural language is also a practical application area for using of the NLC framework. There are similar products in market which are usually called as automotive solutions (Nuance 2013). The main features of these applications are:

- Voice destination entry
- Point-of-interest search
- Street name read out
- Command and control
- Traffic read out
- Location based services.

The language support of these automotive systems are limited, English, Dutch, French, Italian, German, Spanish, Polish, Czech, Danish, Russian and some other European languages are only supported, the Hungarian is not among them mainly because of the hard speech recognition.

The aim of navigation sample application is to show how such system like Google Maps can be extended with natural language control using NLC framework proving its domain-adaptivity property.

6.3.1 Navigation domain knowledge

Since the primary goal of the application is to show the applicability of the framework for several domains, the concept set is not complete, it consists of only several point-of-interests, actions, etc. The knowledge can certainly be extended anytime by adding new nodes into the concept tree.

The following table shows some examples for concepts of the system.

DOI: 10.14750/ME.2013.009

Table 6.4. *Concepts of navigation application*

Concept type	Values
Predicate concepts:	drive, find, go, is, navigate, eat, read, sleep, teach, etc…
Abstract concepts:	restaurant, school, swimming pool, museum, hotel, food, city, etc…
Individual concepts:	Miskolc, Budapest, megyei, MOL, etc…

The concept tree belonging to a predicate concept can be more compound than in case of robot controlling application, since a function here can have more parameters. Figure 6.13 shows the restaurant concept hierarchy which contains the actions, nationalities, objects as well as the central abstract concepts.

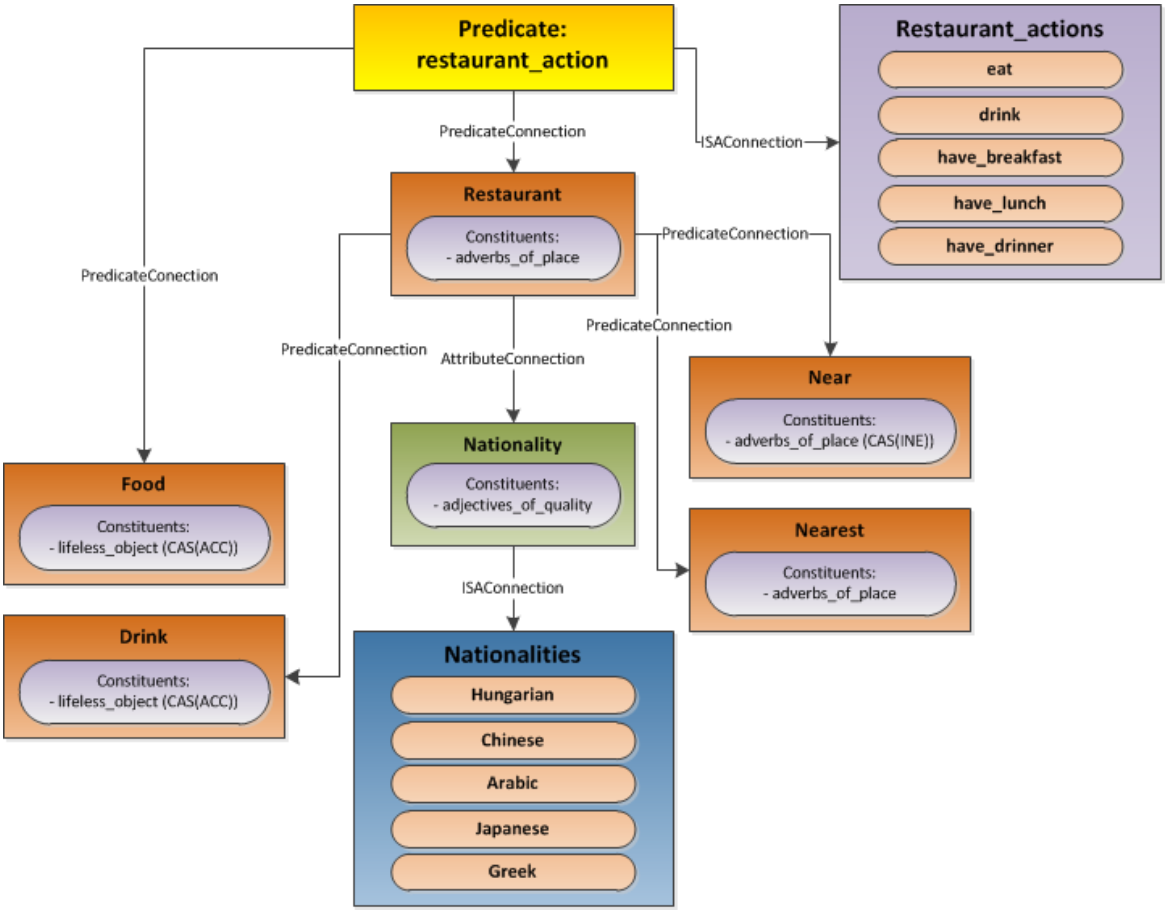

Figure 6.13. *Restaurant concept hierarchy*

6.3.2 Navigation function set

The primary navigation functions are the default in systems like this, but some extra features make the application more user-friendly and interesting. Such functions are e.g. a call number of POI, show route on map dynamically, etc. The following functions are available in our application.

DOI: 10.14750/ME.2013.009

Table 6.5. *Functions of Navigation Application*

Function	Description
Call	User can call the phone number of a POI if it is found in Google Maps places data.
Distance	User can query the distance between two cities, addresses, POI from current position.
Place	User can query location information from POIs.
PlaceData	User can ask for details of POIs.
Position	User can query his/her actual position.
Route	User can get the route information between two endpoints. (city, address, POI, etc…)
Show	User can ask for showing routes, places and distances on the map.
ShowRoute	User can ask for showing route plan on the user interface between two endpoints.

6.3.3 Calling Google Maps services

Google Maps web services can be accessed through HTTP requests. Parameters which are extracted from the natural language text should be added as request parameters. The HTTP answer contains JSON or XML data which describes locations, direction, POI data, etc…

There are four different APIs of Google Maps which are used in the navigation application such as

- **Directions API:** it results the route between two endpoints regarding the mode of transport
- **Distance Matrix API:** it results the distance between two points regarding the mode of transport
- **Geocoding API:** it results the geocoordinates for a given city or address. The reverse geocoding is also possible with this service
- **Places API:** it results point-of-interests round about an origin within a given distance regarding the mode of transport. Places API has a child API called place details which results the detailed information about the given places such as events, phone number, name, email, website, etc.

The application implements the lexical NLP level with the use of a memory structure which remembers the executed commands and can search and bind references in the actual command to the previously performed one. With this kind of trick, a dialog with many sentences can be handled. The dialog is terminated if a new request type arrives. E.g., if the user asks: “How far

DOI: 10.14750/ME.2013.009

is Budapest?", and after he/she says or types "Show on map!" then the application knows what to be visualized.

6.3.4 User interface of navigation application

The structure of user interface is shown in the following figure. There are six main parts of the interface that are highlighted by black circles: actual position, conversation panel, input area, route panel, progress indicator and map panel.

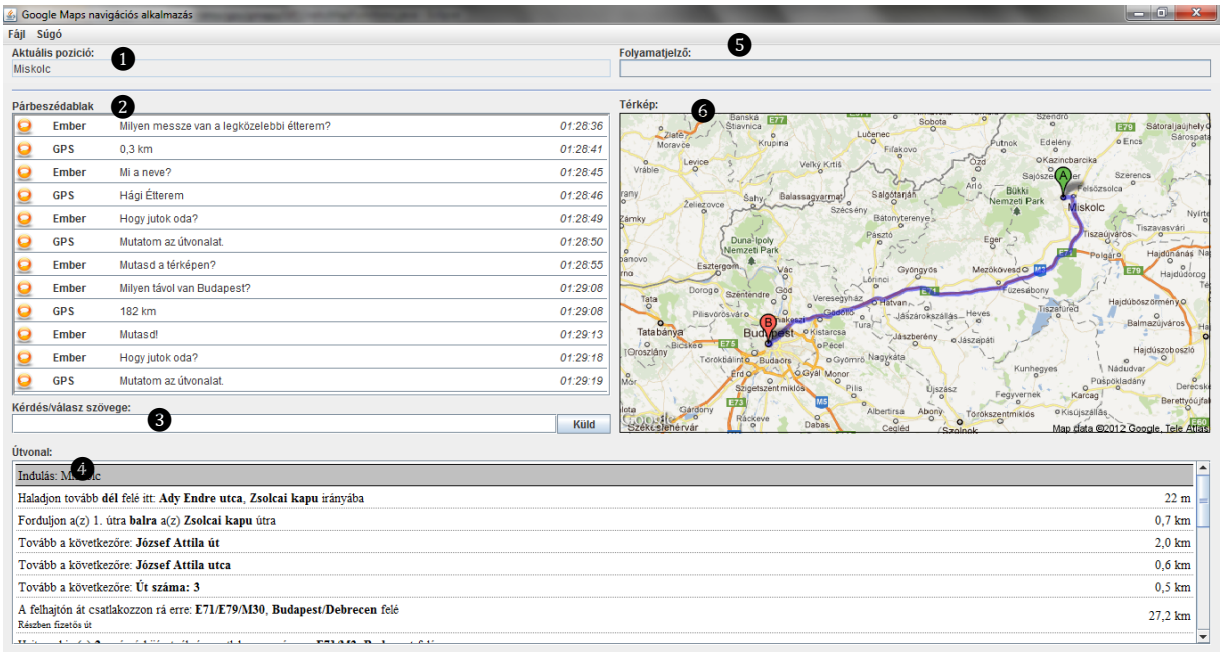

Figure 6.14. *User interface of navigation application*

The actual position region contains the address of our current location. If no GPS locator is used, then it has a constant value which is set in application properties file. It can also be set during installation.

If we use the GPS locator of a smart phone, then if our position has changed, the actual position text area will contain the address calculated from the received latitude and longitude coordinates. This field is read only.

The conversation panel contains the questions, commands and the application answers. A conversation entry consists of four parts,

- The entry type icon, which indicated whether the entry is textual or speech
- The name of the user who has sent the conversation entry. It can be: Human or GPS
- The content of conversation entry in natural language format
- The time of entry when it has been displayed in the panel.

Users can write natural language texts into textbox of the input area. The command can be sent by pressing the *'Enter'* key or pressing the *'Küld'* button. After sending a command, it will be processed and answered by the application.

DOI: 10.14750/ME.2013.009

If the command has missing parameters or the application does not understand the command, a question entry will be shown in conversation panel.

If users query about route plans, like *"Hogy jutok el…"*, a route plan will be provided as primary system answer. The route plan contains the origin and the destination positions and the steps of the route, like in a GPS tool. The route plan contains the distances in the last column, too. The route plan always appears in the route panel. The system writes *"Mutatom az útvonalat."* string into the conversation area to indicate that the primary answer should be found in another panel.

The application uses Google Maps Web Services which are asynchronous, so there can be delay between the question and the answer. To show that something is happening in the background and the application is not frozen, a progress indicator is used and animated till the web service answer arrives.

The most spectacular answer of the application is a map shown in the map panel. The user can query anytime to show a route or the searched POI or POIs on the map. He/she has only to ask *"Mutasd!"* or *"Mutasd a térképen!"*. On the map several things can be shown:

- A POI which has been searched
- Array of POIs if the result contains more items. The limitation can be set in application properties
- A route between two places or cities
- A route towards a searched place from our current position.

The size and rendering of the map depends on the current width and height of the application window. If a map is shown in the panel and the window size is increased, the quality of map will fall off. In next map request, the new map will already be in a good resolution since the size of the map should be sent in the service request.

6.4 Summary of results

In this chapter the proposed framework developed in Java language has been described. In order to demonstrate the theoretical results two sample applications have been implemented using the proposed NLC framework. The new scientific results can be summarized as follows:

Thesis 4.

A prototype Text-to-Function API library was developed to demonstrate the efficiency of the proposed natural language controlling framework. The functionality of the library was tested in different industrial solutions on two different domains (humanoid robot controlling, Google Maps navigation). The experiments show that the implemented systems meet the industrial requirements concerning the accuracy and the response time.

DOI: 10.14750/ME.2013.009

7 Summary

In this dissertation a natural language controlling framework introduced in Figure 3.1 on page 17 is developed. The framework satisfies four major requirements.

1. Domain-adaptivity: the ability to easily learn concepts and relations of different domains without modifying the inner structure or workflows of the framework
2. Language-adaptivity: the ability to parse natural language sentences specified in different languages with only modifying language-dependent parts of framework
3. Extendibility: the ability to extend the set of functions which wanted to be called by natural language commands
4. Open interface: the ability to utilize existing components of NLP engines and to implement, refine any part of the framework for own needs.

Accordingly, the present research has realized the following tasks:

1. The structure of natural language controlling framework had to be created considering domain- and language-adaptivity and combining exiting NLP engines. (see Chapter 3).
2. Semantic models and application function descriptions had to be developed for representing domain knowledge, for sentence analysis, for function mapping and algorithms had to be implemented which perform sentence analysis and function-mapping (see Chapter 4).
3. Optimization problems had to be examined in framework modules and recommendations had to be suggested to achieve execution with lower costs and higher accuracy (see Chapter 5).
4. The framework and two sample applications had to be implemented for the demonstration of theoretical results (see Chapter 6).

7.1 Contributions

The new scientific results which are achieved during the completion of the project are summarized as follows.

Thesis 1.

A novel structure of natural language controlling framework has been developed fulfilling requirements specified in recommendations (R1). The architecture of the framework is based on the developed formal information flow model. The framework contains four modules in a linear structure. The proposed architecture provides domain-adaptivity, language-adaptivity and high extensibility with an open interface. These properties provide high level reusability of the framework in software development in the field of human-machine interfaces.

DOI: 10.14750/ME.2013.009

Thesis 2.

A novel semantic model is developed which satisfies the requirements of the knowledge representation format in our proposed natural language controlling system (R2). The model is an extension of ECG model proposed by (Varga 2011). The implemented extensions (three new types of nodes, three novel edge types completed with morphology and POS labels) ensure a more efficient knowledge transformation between input sentences of a language and the function call generator module. The novelty of the developed function signature model is the inclusion of POS information for function mapping. A deterministic algorithm has been implemented to convert the sentence analysis tree into API function calls.

Thesis 3.

I have determined the key cost components in the natural language controlling framework. The proposed multi-criteria cost model includes execution time, accuracy and resource consumption factors. Algorithm optimization was performed for the POS tagging, sentence analysis and function mapping modules. The proposed graph-based algorithm provides an efficient execution for the implementation of the framework in large-scale domains.

Thesis 4.

A prototype Text-to-Function API library was developed to demonstrate the efficiency of the proposed natural language controlling framework. The functionality of the library was tested in different industrial solutions on two different domains (humanoid robot controlling, Google Maps navigation). The experiments show that the implemented systems meet the industrial requirements concerning the accuracy and the response time.

DOI: 10.14750/ME.2013.009

Reference List

Aldebaran. 2013. “NAO.” Aldebaran. Retrieved June 6, 2013 (http://www.aldebaran-robotics.com).

Alexin, Zoltán, János Csirik, András Kocsor, Márton Miháltz, and György Szarvas 2006. “Construction of the Hungarian EuroWordNet Ontology and its Application to Information Extraction.” Pp. 291-292 in *Proceedings of the Third International Global WordNet Conference (GWC-06)*. Jeju Island, Korea.

Alshawi, H. and J. van Eijck. 1989. *Logical Forms In The Core Language Engine*.

Antoniol, G., B. Caprile, A. Cimatti, R. Fiutem, and G. Lazzari 1994. “Experiencing real-life interaction with the experimental platform of MAIA.” in *Proceedings of the 1st European Workshop on Human Comfort and Security*.

Apache, UIMA. 2013. (http://uima.apache.org).

Asoh, H., T. Matsui, J. Fry, F. Asano, and S. Hayamizu 1999. “A spoken dialog system for a mobile office robot.” Pp. 1139-1142 in *Proc. of Eurospeech'99*. Budapest.

Aust, H. and M. Oerder 1995. “Dialogue control in automatic inquiry systems.” Pp. 121-124 in *Proceedings of the ESCA Workshop on Spoken Dialogue Systems*. Vigso, Denmark.

Baader, F., D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider. 2003. *The Description Logic Handbook: Theory, Implementation, Applications.* Cambridge, UK: Cambridge University Press.

Badler, N. I., B. L. Webber, J. Kalita, and J. Esakov 1991. “Animation from Instructions.” Pp. 51-93 in *Making Them Move: Mechanics, Control, and Animation of Articulaited Figures*. San Mateo, CA: Morgan Kaufmann.

Bechhofer, S., F. van Harmelen, J. Hendler, I. Horrocks, D. McGuinness, P. Patel-Schneider, and L. Stein 2004. “OWL Web Ontology Language Reference, W3C Recommendation.”

Bohnet, B. and J. Niver 2012. “A Transition-Based System for Joint Part-of-Speech Tagging and Labeled Non-Projective Dependency Parsing.” Pp. 1455-1465 EMNLP-CoNLL.

Burgard, W., A.B. Cremers, D. Fox, D. Hahnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun 1998. “The interactive museum tour-guide robot.” in *Proceedings of the Fifteenth National Conference on Artificial Intelligence*. Madison, Wi.

Cannan, J. and H. Hu. 2011. “Human-Machine Interaction (HMI): A survey.” University of Essex, UK.

Chapman, D. 1991. “Vision, Instruction, and Action.” Cambridge, MA: MIT Press.

DOI: 10.14750/ME.2013.009

Chomsky, N. 1957. *Syntactic Structures. The Hague: Mouton & Co*.

Chomsky, N. 1963. *Formal properties of grammar*. MIT Press.

Chomsky, N. 1965. *Aspects of the Theory of Syntax*. Cambridge, MA: MIT Press.

Cullingford, R. 1981. *SAM*.

Csendes, Dóra, János Csirik, Tibor Gyimóthy, and András Kocsor 2005. "The Szeged Treebank." Pp. 123-131 in *Proceedings of the 8th International Conference on Text, Speech and Dialogue (TSD 2005)*. Karlovy Vary, Czech Republic: Springer LNAI 3658.

D'Argenio, P.R., J.P. Katoen, and E. Brinksma 1998. "An algebraic approach to the specification of stochastic systems (extended abstract)." Pp. 126-147 in *Programming Concepts and Methods*. London, UK: Chapman & Hall.

Erdmann, M. 2001. *Ontologien zur konzeptuellen Modllierung der Semantik von XML*. BoD-Books on Demand.

Flippo, F. August 2003. "A Natual Human-Computer Interface for Controlling Wheeled Robotic Vehicles, Thesis." Delft University of Technology: Nederland.

Freitag, D. and A. McCallum 2000. "Information extraction with HMM structures learned by stochastic optimization." in *Proc. AAAI 2000*.

Ghosh, A. and S. Dehuri 2005. "Evolutionary algorithms for multi-criteria optimization: A Survey." in *International journal*.

Ghosh, A., S. Dehuri, and S. & Ghosh. 2008. *Multi-objective evolutionary algorithms for knowledge discovery from databases*. Springer.

Ghosh, A. and B. Nath 2004. "Ghosh, A., & Nath, B. (2004). Multi-objective rule mining using genetic algorithms." Pp. 123-133 in *Information Sciences 163(1).*

Gildea, D. and D. Jurafsky 1995. "Automatic Induction of Finite State Transducer for Simple Phonological Rules." in *Meeting of ACL*.

Goldstein, P. and D. Bobrow. 1980. *A Layered Approach to software Design*. Xerox Publ.

Gruber, T. R. 1995. "Toward principles for the design of ontologies used for knowledge sharing." *International journal of human computer studies* 43:907-928.

Guttman, A. 1984. "R-trees: a dynamic index structure for spatial searching." Pp. 47-57 ACM.

Halácsy, P., A. Kornai, L. Németh, A. Rung, I. Szakadát, and V. Trón 2003. "A szószablya projekt-www.szoszablya.hu." Pp. 298-299 MSZNY 2003: Szeged, Hungary.

Hudson, R. 2007. *Language Networks: The new Word Grammar*. Oxford University Press.

Hulstijn, J., R. Steetskamp, H. ter Doest, S. van de Burgt, and A. Nijholt 1996. "Topics in SCHISMA dialogues." Pp. 89-99 in *Proceedings of the Twente Workshop on Language Technology: Dialogue Management in Natural Language Systems (TWLT 11)*.

DOI: 10.14750/ME.2013.009

Hutchins, W. J. 2004. *The Georgetown-IBM experiment demonstrated in January 1954.* Springer Berlin Heidelberg.

Hutchins, J. 2005. “The history of machine translation in a nutshell.”.

J. R. Pierce, J. B. C. e. a. 1966. “Language and Machines — Computers in Translation and Linguistics. ALPAC report.” Washington, DC.

Joshi, A.K., S. Levy, and M. Takahashi 1975. “Tree adjunct grammars.” Pp. 136-163

Klyne, G. and J. Carroll 2004. “Resource Description Framework (RDF):Concepts and Abstract Syntax.” in *W3C Recommendation*.

Kovács, L. 2004. *Adatbázisok tervezésének és kezelésének módszertana*. Budapest: ComputerBooks.

Kovács, L. 2010. “Rule approximation in metric spaces.” Pp. 49-52 in *Applied Machine Intelligence and Informatics (SAMI), 2010 IEEE 8th International Symposium*. IEEE.

Kovács, L. and T. Sieber 2009. “Multi-layered semantic data models.” Pp. 1130-1135 Hersey (USA): IGI Global Publisher.

Kowalski, R.A. 1974. “Predicate logic as programming language.” Pp. 570-574 in *Proceedings of the IFIPS Congress (Amsterdam)*. International Federation of Information Processing Societies.

Krenn, B. and C. Samuelsson. 1997. *The Linguistic's Guide*.

Krovetz, R. 1993. “Viewing morphology as an inference process.” Pp. 191-203 in *Proceedings of ACM-SIGIR93*.

Kumar, A. 2009. “MONK Project: Architecture Overview.” *Techical Report of the Northwestern University*.

Lafferty, J. D., A. McCallum, and F. C. N. Pereira 2001. “Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data.” Pp. 282-289 in *Proceedings of the Eighteenth International Conference on Machine Learning (ICML 2001)*. Morgan Kaufmann Publishers.

Lee, G., J.H. Lee, H. Rho, Y.T. Park, J. Choi, and J. Seo. 1998. *Interactive NLI agent for multiagent Web search model*. 4th World Congress on Expert Systems.

Lemon, O., A. Bracy, A. Gruenstein, and S. Peters June 2001. “A Multi-Modal Dialogue System for Human-Robot Conversation.” in *Demo, NAACL2001*. Pittsburgh, USA.

Lobin, H. 1992. *Situierte Agenten als natürlichsprachliche Schnittstellen.* Arbeitsberichte Computerlinguistik 3-92: Univ. Bielefeld, Germany.

Lovins, J.B. 1969. *Development of stemming algorithm*. MIT Processing Group, Electronic Systems Laboratory.

DOI: 10.14750/ME.2013.009

MacCartney, B. 2009. “Natural Language Interface.” in *PhD dissertation*. Stanford University.

Manning, C. and H. Schütze. 1999. *Foundations of Statistical Natural Language Processing*. MIT Press.

McCallum, Andrew K. 2002. “MALLET: A Machine Learning for Language Toolkit.” (http://mallet.cs.umass.edu).

McTear, M. 1997. “Spoken Dialogue Technology: Enabling the Conversational User Interface.” in *Distributed at the DUS/ELSNET Bullet Course on Designing and Testing Spoken Dialogue Systems*.

McTear, M. 1998. “Modelling spoken dialogues with state transition diagrams: experiences of the CSLU toolkit.” Pp. 1223-1226 in *Proceedings of the International Conference on Spoken Language Processing*. Sydney, Australia: Australian Speech Science and Technology Association, Incorporated.

McTier, M. 2002. “Spoken dialogue technology: enabling the conversational interface.” Pp. 90-169 in *ACM Computing Surveys*.

Minsky, M. 1975. “A Framework for Representing Knowledge.” in *The Psychology of Computer Vision*. New York, NY: McGraw-Hill.

MorphAdorner. 2009. (http://morphadorner.northwestern.edu).

Németh, L. 2011. “Hunspell: open source spell checking, stemming, morphlogical analysis and generation under GPL, LGPL or MPL licenses.” (http://hunspell.sourceforge.net).

Ng-Thow-Hing, V. and S. O. Pengcheng Luo. 2010. *Synchronized gesture and speech production for humanoid robots*.

Nilsson, N.J. 1984. “Shakey the Robot.” Artificial Intelligence Center: SRI International, Menlo Park, CA.

NLTK. 2012. (http://nltk.org).

Norman, Donald A. 1999. *The Invisible Computer*. MIT Press.

Nuance. 2013. “Nuance - Ingellient systems.” Nuance Communications, Inc. Retrieved June 6, 2013 (http://www.nuance.com).

Pacie, C.D. 1994. “An evaluation method for stemming algorithms.” Pp. 42-50 in *Proceedings of ACM-SIGIR94.*

Paumier, S., F. Liger, G. Vollant, A. Yannacopoulou, and S. Surcin. 2010. *A NLP Engine from the lab to iPhone*.

Perzanowski, D., A. Schultz, Adams W., K. Wauchope, and E., M. B. Marsh June 2001. “Interbot: A Multi-Modal Interface to Mobile Robots.” in *Demo, NAACL2001*. Pittsburgh, USA.

DOI: 10.14750/ME.2013.009

Porter, M. F. 1980. “An algorithm for suffix stripping.” Pp. 130-137 in *Program: electorinc library and information systems, 14(3)*.

Porter, M. F. 2005. “Snowball stemmers and resources page.” Retrieved July 13, 2005 (http://www.snowball.tartarus.org).

Puerta, A.R. 1997. “A model-based interface development environment.” Pp. 40-47 in *Software, IEEE, vol. 14*.

Quillian, M. 1968. “Semantic Memory.” Pp. 216-270 in *Semantic Information Processing*. Cambridge, MA: MIT Press.

Robinson, J. 1970. “Dependency structures and transformation rules.” Pp. 259-285 in *Language, 46.*

Rucklidge, W. 1996. “Efficient visual recognition using the Hausdorff distance.” Berlin Heidelberg: Springer.

Sato, T. and S. Hirai 1987. “Language-Aided Robotic Teleoperation System (LARTS) for Advanced Teleoperation.” Pp. 476-480 in *IEEE Journal on Robotics and Automation (RA)*.

Schmidt, Albrecht 2002. “Ubiquitous Computing - Computing in Context.” in *PhD thesis*. Lancester: Computing Department, Lancester Univeristy.

Sha, F., P. F. 2003. “Shallow parsing with conditional random fields.” Pp. 134-141 in *Proc. NAACL '03 Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*.

Shank, R. C. 1975. “Conceptual Information Processing.” North Holland, Amsterdam.

Sharma, V., P. Jalote, and K. Trivedi. 2005. *Evaluating Performance Attributes of Layered Software Architecture, Proc. of CBSE 2005, LNCS*.

Sondheimer, N.K. 1976. “Spatial Reference and Natural Language Machine Control.” Pp. 329-336 in *Int. Journal of Man-Machine Studies*.

Sowa, J. 1992. “Semantic Networks.” in *Encyclopedia of Arti*. Wiley.

Sowa, J. 2000. *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Pacific Grove, CA: Brooks Cole Publishing Co.

StanfordNLP. 2013. “Standford NLP.” (http://www-nlp.stanford.edu).

Strassel, S., D. Adams, H. Goldberg, J. Herr, R. Keesing, D. Oblinger, H. Simpson, R. Schrag, and J. Wright. 2010. *The darpa machine reading program-encouraging linguistic and reasoning research with a series of reading tasks*.

Sutton, C. and A Mccallum 2012. “An Introduction to Conditional Random Fields, Foundation and Trends in Machine Learning.” Now Publishers.

DOI: 10.14750/ME.2013.009

Szarvas, Gy., R. Farkas, and A.: Kocsor. 2006. *A Multilingual Named Entity Recognition System Using Boosting and C4.5 Decision Tree Learning Algorithms*. The Ninth International Conference on Discovery Science.

Tesniere, L. 1959. *Elements de syntaxe structurale*. Paris, Klincksieck.

Tordai, A. and M. de Rijke. 2005. *Hungarian Monolingual Retrieval at CLEF*.

Torrance, M.C. 1994. "Natural Communication with Robots. Master's thesis." MIT, Department of Electrical Engineering and Computer Science: Cambridge, MA.

Trón, V., P. Halácsy, P. Rebrus, A. Rung, E. Simon, and P. Vajda. 2006. *morphdb.hu: Hungarian lexical database and morphological grammar*.

Turing, A.M. 1950. *Computing machinery and intelligence*.

Varga, Erika B. 2011. "Ontology-based Semantic Annotation and Knowledge Representation in a Grammar Induction System." in *Ph.D. Dissertation*. Miskolc.

Veldhuijzen van Zanten, G. 1996. "Pragmatic interpretation and dialogue management in spoken-language systems." Pp. 81-88 in *Dialogue Management in Natural Language Systems, TWLT11*. University of Twente.

Vere, S. and T. Bickmore 1990. "A Basic Agent." Pp. 41-60 in *Computational Intelligence*.

Versweyveld, L. March 1998. "Voice-controlled surgical robot ready to assist in minimally invasive heart surgery." in *Virtual Medical Worlds Monthly*.

Wei, L. and H. Hu. 2011. "Towards Multimodal Human-Machine Interface for Hands-free Control: A survey." University of Essex, UK.

Weiner, P. 1973. "Linear pattern matching algorithms." Pp. 1-11 in *Switching and Automata Theory*. SWAT'08. IEEE Conference Record of 14th Annual Symposium: IEEE.

Weiser, M. 1993. "Hot topics: Ubiquitous computing." in *IEEE Computer*.

Weiser, M. and J.S. Brown 1998. "The coming age of calm technology." Pp. 75-85. in *Beyond calculation: The next fifty years of computing*. New York, NY.

Weizenbaum, J. 1966. *ELIZA-A Computer Program for the study of Natural Language Communication between man and machines*.

Wejchert, Jakub 2000. ""The Disappearing Computer"." in *Information Document, IST Call for proposals*. European Commission, Future and Emerging Technologies.

Winograd, Terry. 1972. *Understanding Natural Language*. New York: Academic Press.

Zhao, Y. and Y. Yao 2006. "Classification based on logical concept analysis." Pp. 419-430 in *Advances in Artifical Intellihence*. Springer Berlin, Heidelberg.

DOI: 10.14750/ME.2013.009

Zsibrita, János, Veronika Vincze, and Richárd Farkas 2013. “magyarlanc 2.0: szintaktikai elemzés és felgyorsított szófaji egyértelműsítés.” Pp. 368-374 in *IX. Magyar Számítógépes Nyelvészeti Konferencia.* Szeged, Szegedi Tudományegyetem.

DOI: 10.14750/ME.2013.009

Author's publications

CHAPTERS IN FOREIGN-LANGUAGE PUBLISHED ABROAD

[1] **Kovács, L., Barabás, P., Répási, T.:** *Ontology-Based Semantic Models for Databases*, Handbook of Research on Innovations in Database Technologies and Applications: Current and Future Trends, IGI Global Publisher, Hersey (USA) 2009, ISBN 978-1-60566-242-8, Pp. 443-451

REVIEWED PUBLICATIONS IN FOREIGN-LANGUAGE PUBLISHED IN INTERNATIONAL JOURNAL

[2] **Barabás, P., Kovács, L.:** *Optimization tasks in Conversion of Natural Language Text into Function Calls,* Topics in Intelligent Engineering and Informatics, ISSN: 2193-9411, Springer, 2013

[3] **Barabás, P., Kovács, L.:** *Efficient Encoding of Inflection Rules in NLP Sytems*, Scientific Bulletin, vol. 9 (XXVI), no. 2, 2012, ISSN 2285-438X, Pp. 11-16

REVIEWED PUBLICATIONS IN FOREIGN-LANGUAGE PUBLISHED IN DOMESTIC JOURNAL

[4] **Barabás, P., Kovács, L.:** *Estimation of Misclassification Error using Bayesian Classifier*, Publication of University of Miskolc, Production Systems and Information Engineering , Vol. 5, 2009, ISSN 1785-1270, Pp. 41-50.

[5] **Barabás, P., Kovács, L.:** *Efficient Classification of String Transformations using Markov Model*, GAMF Közlemények , XXI. évf., 2008, ISSN-1587-4400, Pp. 145-151

DOI: 10.14750/ME.2013.009

REVIEWED PUBLICATIONS IN HUNGARIAN LANGUAGE PUBLISHED IN DOMESTIC JOURNAL

[6] **Barabás Péter:** *Parancskinyerés magyar nyelvű szövegből*, A Gépipari Tudományos Egyesület Műszaki Folyóirata, LXIII. Évfolyam, 2012, pp. 71-74.

REVIEWED PUBLICATIONS IN FOREIGN-LANGUAGE PUBLISHED IN INTERNATIONAL CONFERENCE PROCEEDINGS

[7] **Barabás, P., Kovács, L., Vircikova, M.:** *Robot Controlling in Natural Language*, The 3rd IEEE International Conference on Cognitive Infocommunications (CogInfoCom2012), Kosice, Slovakia, December 2-5, 2012, Pp. 181-186

[8] **Barabás, P., Kovács, L.:** *Requirement Analysis of Internal Modules of Natural Language Processing Engines*, 10th International Symposium on Application Machine Intelligence and Informatics, Her'lany (Slovakia) 2012, ISBN 978-1-4577-0196-2, pp. 41-46

[9] **Kovács, L., Barabás, P.:** *Experiences of building of context-free grammar tree*, 9th International Symposium on Application Machine Intelligence and Informatics, Smolenice (Slovakia) 2011, ISBN 978-1-4244-7429-5, pp. 67-71

[10] **Barabás, P., Kovács, L.:** *Implementation of Sentence Parser for Hungarian Language in Natural Language Processing*, 8th International Symposium on Applied Machine Intelligence and Informatics, Herl'any, Slovakia, 1/2010, ISBN 978-1-4244-6422-7, pp. 59-63

[11] **Kovács, L., Baksa-Varga, E., Répási, T., Barabás, P.:** *Clustering Based on Context Similarity*, Complexity and Intelligence of the Artificial and Natural Complex Systems, IEEE computer Society, ISBN 139780769536217, 2009., pp. 157-166

[12] **Kovács, L., Barabás, P.:** *Generalization Analysis of the CL and MM-based Classifications,* 6th International Symposium on Applied Machine Intelligence and Informatics, Herl'any (Slovakia) 2008, ISBN 978-1-4244-2105-3, pp. 39-43.

DOI: 10.14750/ME.2013.009

NON-REVIEWED PUBLICATIONS IN FOREIGN-LANGUAGE PUBLISHED IN INTERNATIONAL CONFERENCE PROCEEDINGS

[13] **Barabás, P., Kovács, L.:** *Usability of Summation Hack in Bayesian Classification*, 9th International Symposium of Hungarian Researchers on Computational Intelligence and Informatics, Budapest 2008

[14] **Barabás, P.:** *Rule Learning with MM-based Classification*, MicroCAD 2008 International Scientific Conference, Miskolc, 03/2008

[15] **Kovács, L., Barabás, P.:** *Cost Analysis of Classification using CL and MM*, 8th International Symposium of Hungarian Researchers on Computational Intelligence and Informatics, Budapest, 2007., pp. 227-237.

[16] **Kovács, L., Barabás, P.:** *Statistical Methods for Morhological Parsers*, 7th International Symposium of Hungarian Researchers on Computational Intelligence and Informatics, Budapest, 2006, pp. 523-531.

[17] **Barabás, P.:** *Automated* Type Checking in VFP, MicroCAD 2005 International Scientific Conference, Miskolc, pp. 229-234

NON-REVIEWED PUBLICATIONS IN FOREIGN-LANGUAGE PUBLISHED IN LOCAL CONFERENCE PROCEEDINGS

[18] **Barabás, P.:** *Cost Analysis of Classification using (H)MM*, Forum of PhD Students, Miskolc, 11/2007

[19] **Barabás, P.:** *Automations in Grammar Induction*, Forum of PhD Students, Miskolc, 11/2006

DOI: 10.14750/ME.2013.009

Appendix A

POS LABEL SYSTEM

POS labels of stems

Label	Description
ADJ	Adjective
ADV	Adverb
ART	Article
CONJ	Conjunction
DET	Determiner
NOUN	Noun
NUM	Numeral
ONO	Onomatopoeic
POSTP	Postposition
PREP	Preposition
PREV	Preverb
UTT-INT	Interjection
VERB	Verb

POS labels of nouns

Property	Value	Label
Number	Singular	-
	Plural (simple)	<PLUR>
	Plural (familiar)	<PLUR<FAM>>
Possessor	None	-
	1st person	<POSS<1>>
	2nd person	<POSS<2>>
	3rd person	<POSS>

DOI: 10.14750/ME.2013.009

	Singular Plural	<POSS> <POSS<PLUR>>
Possessed	None Singular Plural	- <ANP> <ANP<PLUR>>
Case	NOM ACC (-t) DAT (-nak, nek) INS (-val, -vel) CAU (-ért) TRA (-vá, -vé) SUE (-on,-en, -ön) SBL (-ra, -re) DEL (-ról, -ről) INE (-ban, -ben) ELA (-ból, ből) ILL (-ba, -be) ADE (-nál, nél) ALL (-hoz, -hez, -höz) ABL (-tól, -től) TER (-ig) FOR (-ként)	- <CAS<ACC>> <CAS<DAT>> <CAS<INS>> <CAS<CAU>> <CAS<TRA>> <CAS<SUE>> <CAS<SBL>> <CAS<DEL>> <CAS<INE>> <CAS<ELA>> <CAS<ILL>> <CAS<ADE>> <CAS<ALL>> <CAS<ABL>> <CAS<TER>> <CAS<FOR>>

POS labels of verbs

Property	Value	Label
Modality	None Modal	- <MODAL>
Mood	Conjunctive Subjucnctive/Imperative Conditional	- <SUBJUNC-IMP> <COND>
Tense	Present Past	- <PAST>

DOI: 10.14750/ME.2013.009

	Future	<FUT>
Number	Singular Plural	- <PLUR>
Person	1st 1st with second person object 2nd 3rd	<PERS<1>> <PERS<1<OBJ<2>> <PERS<2>> <PERS>
Definiteness	Indefinite Definite	- <DEF>

Labels for derivational morphemes

Explanation	Suffix value	Label	Source POS	Target POS
Frequentative	-gat, -get	FREQ	VERB	VERB
Medial	-ódik, -ődik	MEDIAL	VERB	VERB
Causative	-tat, -tet	CAUS	VERB	VERB
Adverbial participle	-va, -ve	PART	VERB	ADV
Perfect adverbial participle	-ván, -vén	PERF_PART	VERB	ADV
Imperfect adverbial participle	-ó, -ő	IMPERF_PART	VERB	ADJ
Future adjectival participle	-andó, -endő	FUT_PART	VERB	ADJ
Perfect adjectival participle	-ott	PERF_PART	VERB	ADJ
Negative perfect adjectival participle	-atlan, -etlen	NEG_PERF_PART	VERB	ADJ
Gerund	-ás, -és	GERUND	VERB	NOUN
Negative modal adjectival participle	-hatatlen, -hetetlen	NEG_MODAL_PART	VERB	ADJ
Modal adjectival participle	-ható, -hető	MODAL_PART	VERB	ADJ
Regular activity	-kodik, -ködik	REG_ACT	NOUN	VERB
Abstract	-ság, -ség	ABSTRACT	NOUN	NOUN
Mrs	-né	MRS	NOUN	NOUN
Diminutive	-ka, -ke	DIMIN	NOUN	NOUN
Attributive	-s	ATTRIB	NOUN	ADJ

DOI: 10.14750/ME.2013.009

Metonymical attributive	-i	MET_ATTRIB	NOUN	ADJ
Inalienable attributive	-jú, jű	INAL_ATTRIB	NOUN	ADJ
Negative attributive	-talan, -telen	NEG_ATTRIB	NOUN	ADJ
Negative attributive2	-mentes	NEG_ATTTIB2	NOUN	ADJ
Type1	-szerű	TYPE1	NOUN	ADJ
Type2	-féle	TYPE2	NOUN	ADJ
Type3	-nemű	TYPE3	NOUN	ADJ
Type4	-fajta	TYPE4	NOUN	ADJ
Type rank	-rangú	TYPE_RANK	NOUN	ADJ
Locative inessive	-beli	LOC_INE	NOUN	ADJ
Quantity	-nyi	QUANTITY	NOUN	NUM
Essivus formalis	-képpen	ESS_FOR	NOUN	ADV
Comitative	-stul, -stül	COM	NOUN	ADV
Period1	-anként	PERIOD1	NOUN	ADV
Period2	-onta, -ente	PERIOD2	NOUN	ADV
Activity1	-oz, -ez, -öz	ACT	NOUN	VERB
Activity2	-ol, -el, -öl, -ül	ACT2	NOUN	VERB
Comparative	-bb	COMPAR	ADJ	ADJ
Superlative	Leg-bb	SUPERLAT	ADJ	ADJ
Supersuperlative	Legesleg-bb	SUPERSUPERLAT	ADJ	ADJ
Comparative designative	-bbik	COMPAR_DESIGN	ADJ	ADJ
Superlative designative	Leg-bbik	SUPERLAT_DESIGN	ADJ	ADJ
Supersuperlative designative	Legesleg-bbik	SUPERSUPERLAT_DESIGN	ADJ	ADJ
Manner	-lag	MANNER	ADJ	ADV
Manner	-an, -án, -en, -én	MANNER	ADJ	ADV
Intransitive resultative	-odik, -edik, -ul, -ül	INTRANS_RESULT	ADJ	VERB
Transitive resultative	-ít	TRANS_RESULT	ADJ	VERB
Multiplicative iterative	-szor, -szer, -ször	MULTIPL_ITER	NUM	ADV
Multiplicative iterative	-szoroz, -szerez, -szöröz	MULTIPL_ITER	NUM	VERB
Iterative attributive	-szori, -szeri, -szöri	ITER_ATTRIB	NUM	ADJ

DOI: 10.14750/ME.2013.009

Multiplicative attributive	-szoros, -szeres, -szörös	MULTIPL_ATTRIB	NUM	ADJ
Multiplicative	-szorta, -szerte, -szörte	MULTIPL	NUM	ADV
Aggregative	-an, -en	AGGREG	NUM	ADV
Fractional	-ad, -ed, -öd	FRACT	NUM	NUM
Ordinal	-odik, -edik, -adik, -ödik	ORD	NUM	NUM
Date	-odika, -edike, -adika, -ödike	DATE	NUM	NOUN
Attributive	-i	ATTRIB	POSTP	ADJ

DOI: 10.14750/ME.2013.009

Appendix B

XML SCHEMA DEFINITIONS

XSD 0.1. Definition of text parser output

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://hu/miskolc/uni/iit/nli/text"
            xmlns="http://hu/miskolc/uni/iit/nli/text"
            elementFormDefault="qualified"
<xs:complexType name="Word">
  <xs:complexContent>
    <xs:restriction base="xs:string">
       <xs:attribute name="entity" type="xs:string"/>
       <xs:attribute name="suffix" type="xs:string"/>
    <xs:restriction/>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="Sentence">
  <xs:sequence>
    <xs:element name="Content" type="xs:string"/>
    <xs:element name="Words">
      <xs:complexType>
         <xs:sequence>
           <xs:element name="Word" type="xs:string" maxOccurs="unbounded"/>
         </xs:sequence>
       </xs:complexType>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="type" use="required">
    <xs:simpleType>
       <xs:restriction base="xs:string">
         <xs:enumeration value="declarative"/>
         <xs:enumeration value="interrogative"/>
         <xs:enumeration value="imperative"/>
       </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
</xs:complexType>
<xs:element name="TextParserOutput">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Sentence" type="Sentence" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

XSD 0.2. Definition of morphology output

DOI: 10.14750/ME.2013.009

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://hu/miskolc/uni/iit/nli/morpho"
            xmlns="http://hu/miskolc/uni/iit/nli/morpho"
            elementFormDefault="qualified">

<xs:complexType name="Suffixes">
  <xs:sequence>
    <xs:element name="Suffix" minOccurs="0" maxOccurs="unbounded">
      <xs:complexType>
        <xs:simpleContent>
          <xs:extension base="xs:string">
            <xs:attribute name="code" type="xs:string" use="required"/>
          </xs:extension>
        </xs:simpleContent>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="Case">
  <xs:sequence>
    <xs:element name="Stem">
      <xs:complexType>
        <xs:simpleContent>
          <xs:extension base="xs:string">
            <xs:attribute name="class" use="required"/>
          </xs:extension>
        </xs:simpleContent>
      </xs:complexType>
    </xs:element>
    <xs:element name="Suffixes" type="Suffixes"/>
  </xs:sequence>
  <xs:attribute name="id" type="xs:positiveInteger" use="required"/>
</xs:complexType>
<xs:complexType name="AnalysedWord">
  <xs:sequence>
    <xs:element name="Case" type="Case" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="id" type="xs:string" use="required"/>
</xs:complexType>
<xs:complexType name="Sentence">
  <xs:sequence>
    <xs:element name="Content" type="xs:string"/>
    <xs:element name="Words">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="Word" type="AnalysedWord" maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="type" use="required">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="declarative"/>
        <xs:enumeration value="interrogative"/>
        <xs:enumeration value="imperative"/>
      </xs:restriction>
```

DOI: 10.14750/ME.2013.009

```
    </xs:simpleType>
  </xs:attribute>
</xs:complexType>
<xs:element name="MorphoOutput">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Sentence" type="Sentence" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

XSD 0.3. Definition of sentence analysis

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:local="http://hu/miskolc/uni/iit/nli/domain"
       xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://hu/miskolc/uni/iit/nli/domain"
       elementFormDefault="qualified">
       <xs:complexType name="ConceptType">
              <xs:sequence>
                     <xs:element name="Concept" type="local:ConceptType"
                            minOccurs="0" maxOccurs="unbounded" />
              </xs:sequence>
              <xs:attribute name="type" type="xs:anyURI" use="required" />
              <xs:attribute name="value" type="xs:anyURI" />
              <xs:attribute name="usage" type="xs:string" />
       </xs:complexType>
       <xs:element name="SentenceAnalysis">
              <xs:complexType>
                     <xs:sequence>
                            <xs:element name="Concept" type="local:ConceptType"
                                   maxOccurs="unbounded" />
                     </xs:sequence>
              </xs:complexType>
       </xs:element>
</xs:schema>
```

DOI: 10.14750/ME.2013.009