Programming languages for autopoiesis facilitating semantic wiki systems

Schatten, Markus

Doctoral thesis / Disertacija

2010

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Organization and Informatics Varaždin / Sveučilište u Zagrebu, Fakultet organizacije i informatike Varaždin**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:211:648175

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-04-25**

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Repository / Repozitorij:

Faculty of Organization and Informatics - Digital Repository

DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

PODACI O DISERTACIJI

I. AUTOR

Ime i prezime	Markus Schatten
Datum i mjesto rođenja	27. rujan 1981., Beč (Wien), Austrija
Naziv fakulteta i datum diplomiranja na VII/1 stupnju	Fakultet organizacije i informatike, Varaždin, 22. studeni 2005.
Naziv fakulteta i datum diplomiranja na VII/2 stupnju	Fakultet organizacije i informatike, Varaždin, 31. siječanj 2008.
Sadašnje zaposlenje	Fakultet organizacije i informatike, Varaždin, asistent

II. DISERTACIJA

Naslov	Programming Languages for Autopoiesis Facilitating Semantic Wiki Systems
Broj stranica, slika, tablica, priloga, bibliografskih podataka	300 stranica, 46 slika, 3 tablica, 5 priloga, 116 bibliografskih podataka
Znanstveno područje, smjer i disciplina iz kojeg je postignut akademski stupanj	Društvene znanosti, Informacijske znanosti, -
Mentor i sumentor rada	Prof. dr. sc. Mirko Čubrilo Prof. dr. sc. Miroslav Bača
Fakultet na kojem je rad obranjen	Fakultet organizacije i informatike
Oznaka i redni broj rada	

III. OCJENA I OBRANA

Datum prihvaćanja teme od Znanstveno-nastavnog vijeća	18. studeni 2008.
Datum predaje rada	4. lipanj 2009.
Datum sjednice ZNV-a na kojoj je prihvaćena pozitivna ocjena rada	19. siječanj 2010.
Sastav Povjerenstva koje je rad ocijenilo	Prof. dr. sc. Mirko Maleković Prof. dr. sc. Mirko Čubrilo Prof. dr. sc. Miroslav Bača Prof. dr. sc. Michael Kifer Prof. dr. sc. Vladimir Mateljan
Datum obrane rada	
Sastav Povjerenstva pred kojim je rad obranjen	

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE

Markus Schatten

Programming Languages for Autopoiesis Facilitating Semantic Wiki Systems

- DOKTORSKA DISERTACIJA -

VARAŽDIN, 2009

Marineli

Acknowledgements

The author would like to acknowledge the intensive help of his mentor prof. dr. sc. Mirko Čubrilo especially in the field of language formalization. Without him this thesis would probably never had been finished. Also the author would like to acknowledge his co-mentor prof. dr. sc. Miroslav Bača for urging him to new and new scientific ideas and concepts. A special thanks goes to prof. dr. sc. Mirko Maleković for very usefull discussions on autopoietic theory as well as semantic modelling. He would also like to acknowledge prof. Michael Kifer for very useful critics and suggestions, which made this thesis a much better one. Additionally a great thank you should be given here to friend and colleague Bernardo Golenja, mag. inf. the chief of the computer support center at the Faculty of Organization and Informatics. Thank you for not killing me when ┬AOPĪs went wild on arka.foi.hr. Another great thank you is for friend and colleague Gordan Ponjavić who is one of the perpetrators that autopoietic theory became a part of my life. Additionally my friend and colleague prof. dr. sc. Miroslav Žugaj who was the first to introduce the author to scientific work should be acknowledged here. I would also like to thank collegue Jurica Ševa, dipl. inf. for support and discussions. The author would especially like to thank his parents Peter and Branka for their patience and understanding as well as his brother Martin and his wife Ivana and the little ones Lara and Mauro. You brought a lot of joy into my life! The last but greatest thank you goes to my fiancee Marinela. This thesis is dedicated to you. Thank you for all the discussions, all the patience and all the support any men in the world would love to get! <3

Markus Schatten

Fakultet organizacije i informatike

June 2009

Contents

Acknowledgements **iv**

List of Figures **ix**

Chapter 1 Introduction **1**

1.1 Objectives and Hypotheses 4

1.2 Methodology and Framework 5

1.3 Related Work 7

Chapter 2 Wiki Systems **9**

2.1 A Short History of Wiki's 9

2.2 Important Concepts 10

Chapter 3 Languages for Wiki Systems **13**

3.1 Language 13

3.2 Regular Expressions 15

3.3 Grammars 17

3.4 Wiki Syntax 19

3.4.1 Hyperlinks 20

3.4.2 Images and other Objects 23

3.4.3 Headings and Text Formatting 26

3.4.4 Comments 28

3.4.5 Tables 29

3.4.6 Variables and Templates 34

3.4.7 References 34

Chapter 4 From SM over SW to SWS **40**

4.1 Semantic Modeling and the Object - oriented Paradigm 41

4.1.1 Domains . . . 41
4.1.2 Concepts . . . 42
4.1.3 Generalization and Specialization . . . 43
4.1.4 Objects . . . 44
4.1.5 Relations and Mappings . . . 45
4.1.6 Attributes and Attribute Values . . . 46
4.2 The Semantic Web . . . 46
4.2.1 Ontologies . . . 47
4.2.2 Semantic Web Languages . . . 47
4.3 Web Services . . . 49
4.4 Semantic Web Services . . . 49
4.5 Semantic Wiki Systems . . . 50

Chapter 5 Programming Languages for Semantic Wiki Systems 51
5.1 Frame Logic . . . 51
5.2 Semantic Wiki Language . . . 53
5.2.1 Semantic Wiki Syntax . . . 57
5.2.2 Semantic Templates . . . 58
5.2.3 Queries . . . 59
5.2.4 Meta Information . . . 71
5.3 Inconsistencies in Semantic Wiki Systems . . . 72

Chapter 6 Autopoiesis and Autopoietic Systems 74
6.1 Introducing Autopoietic Systems . . . 74
6.2 Various Aspects of Autopoiesis . . . 75
6.3 Invitations to an New Paradigm . . . 78
6.3.1 Heterarchies and the Fishnet Organization . . . 79
6.3.2 Process and Project Oriented Approaches and the Hypertext Organization . . . 80
6.3.3 Organizational Suprastructures and the Virtual Organization . . . 81
6.3.4 Organizational Architecture . . . 82
6.3.5 The Fractal Company . . . 83
6.4 Relations between Social, Organizational and Information Systems . . . 84
6.5 A Critical Review of Autopoiesis . . . 85
6.6 Defining Autopoietic Information Systems . . . 89

6.7 Modern Information and Communication Technologies 92
6.8 Current System Model 95
6.9 Experiences and Lessons Learned 98
6.10 Conclusion 99

Chapter 7 Programming Languages for Autopoiesis Facilitating Semantic Wiki Systems 100

7.1 Social Network Analysis 100
7.1.1 Graph Theory 101
7.2 Probability Annotation 104
7.2.1 Query Execution 105
7.2.2 Query Execution with User-Defined Rules 107
7.3 Annotated Semantic Wiki Language 108
7.4 Amalgamation 120
7.5 Amalgamated Annotated Semantic Wiki Language 121

Chapter 8 The Niklas Language 126

8.1 Wiki Component 127
8.1.1 Hyperlinks 127
8.1.2 Images and Other Objects 127
8.1.3 Headings 128
8.1.4 Text Formatting 128
8.1.5 Lists and Tables 129
8.1.6 Templates and Inclusion 130
8.1.7 References 130
8.2 Semantic Component 131
8.2.1 Class Hierarchies 132
8.2.2 Dictionaries 133
8.2.3 Frequently Asked Questions 134
8.2.4 Tables of Content 134
8.2.5 Who Edited this Page 135
8.2.6 Issue Tracking 136
8.2.7 What Links Here 137
8.3 Autopoietic Component 137
8.3.1 Probability Annotation 137

8.3.2 Amalgamation 138
8.4 A Short Comparison to other Semantic Wiki Engines 139

Chapter 9 Application Examples **145**
9.1 Autopoietic System for Personal Computer Security 145
9.2 Autopoietic Scientific Publishing System 152
9.3 Autopoietic Knowledge Management System 162
9.4 Other Examples of Possible Applications 167

Chapter 10 Conclusion **169**

Bibliography **172**

Appendix A Wiki Parser for niKlas in XSB Prolog **185**

Appendix B Semantic Wiki Parser for niKlas in XSB Prolog **197**

Appendix C Amalgamated Annotated Semantic Wiki Parser for niKlas in XSB Prolog **228**

Appendix D Annotated Query Execution Engine Implementation Issues **262**

Appendix E TaOPis Source Code **270**

Sažetak **282**

Curriculum Vitae **290**

List of Figures

2.1 Important concepts in wiki systems [70] 11

4.1 Concepts as filters in our perception [58] 42
4.2 The concept triad [58] 43
4.3 Generalization and specialization [58] 44
4.4 Dynamic classification [58] 45
4.5 Relations and mappings [58] 46
4.6 Semantic Web Stack [10] 47

5.1 List of examples generated by a query 65
5.2 Tag cloud of programming languages generated by a query 71
5.3 Inconsistent definition of subclasses 72
5.4 Suggestion mechanism entry form 73

6.1 Structural Coupling [75] 78
6.2 The basic autopoietic system 79
6.3 The fishnet organization [37] 80
6.4 The hypertext organization [69] 81
6.5 The virtual organization [7] 82
6.6 Basic concepts of organizational architecture [105] 83
6.7 a. The Mandelbrot fractal, b. A fern twig [105] 84
6.8 The fractal principle [105] 85
6.9 The information system as a subsystem of an organization [13] 86
6.10 Relationships between the social system, organizations, information systems and ICT (Adapted partially from [44]) 87
6.11 The basic (evolving) autopoietic system 88

7.1 Social network of “Pepperland” 112

7.2 Social network of "Yellow submarine" 123
7.3 The integration of two social networks 125

8.1 Predefined class hierarchy in ᴛᴀOPīs 126

9.1 Ranks of the SecureAIS semantic wiki project members 146
9.2 UML diagram of SecureAIS 147
9.3 Adding the first patch to SecureAIS 147
9.4 Wiki page of the first patch on SecureAIS 148
9.5 Adding the second patch to SecureAIS 148
9.6 Attacker adding virus to SecureAIS 150
9.7 Malicious patch wiki page 150
9.8 Query with malicious patches filtered out 151
9.9 Frontpage of JoPoP 153
9.10 Submitting a new manuscript 154
9.11 A manuscript on JoPoP 155
9.12 Multimedia on JoPoP manuscript 155
9.13 Tags on a sample bibliographic entry 156
9.14 References generated by a query 157
9.15 A tagged review 159
9.16 List of query generated reviews 160
9.17 List of employees working on more than four projects 165

E.1 ᴛᴀOPīs system's architecture 270

Chapter 1

Introduction

Wiki systems, a progressive technology that hasn't been predicted a bright future by prominent professionals, are in wide usage today. These Web based systems, that allow any user to add different content to the system, are autopoietically evolving into more and more impressive knowledge repositories. Maybe the best known example of such, Wikipedia, the free Internet encyclopedia, had over 2.2 million articles in its English version at the time of writing this text, whilst there are versions for almost all world languages.

Still, it seems that wiki systems came to their edge [103]. It is often the case that various rules concerning behavior, knowledge organization as well as meta data are implemented in order to facilitate search and reasoning in these often huge (mostly textual) data repositories [18, 71, 81, 45].

Efforts like semantic wiki systems, that try to add a semantic component to traditional wiki systems, often ignore one of the most important success factors of wikis. Wiki systems are easy to use, and thus used by a wide spectrum of different people with different knowledge of information technologies - from excellent IT professional over average Internet and computer users to laymans. Obviously the distribution of users tends towards the less conversant in information technologies. This seems to be the main reason why the introduction of advanced technologies like the semantic web greatly limits their ease of use, since the average user needs to have fairly good knowledge of such technologies [84].

As mentioned previously, wiki systems evolve due to autopoiesis of the social system surrounding them, as opposed to traditional alopoietic (technical) application systems. Wiki systems can be explained through the fact that users by participating on the system, (re-)create the system, extend and amplify it with more and more new

content, rules, definitions etc. Thus acquired formalized content is the result of a social systems structural coupling[1]. So a question to answer here is: *is it possible to implement the concept of the semantic web into wiki systems by maintaining their initial ease of use?*

Another type of evolving systems that we like to point out here are social tagging systems [94]. Such systems are in wide use today, especially for personal information and knowledge management (PIM, PKM). These systems allow their users to tag any content they encounter on the Web. They are interesting from a Web search engine perspective due to their impressive results. While common search engines use advanced algorithms to gain meta data, social tagging systems simply use the tags their users created. Due to the well known Delphi effect which states that the average opinion of some subset of a population is a better predictor than the opinion of a randomly chosen person [77], such systems often yield better results. One could say that such systems take advantage of a "collective intelligence", since meta data provided through tags represents the preprocessed original content for personal knowledge organization of individuals.

To put this research into context we shall ask yet another question. Modern organizations today are open, virtual, adaptive, heterarchic and virtual [105]. These fact let us seriously consider that common (rigid, alopoietic) information systems aren't able to support such dynamic organizations [8]. *Could we use modern information technology to support such organizational needs?*

In order to make a first step towards the answer of these questions, we shall introduce a new concept into wiki systems. Web services, are a relatively new technology, that allows the use of remote procedures from all over the Web as if they were local. Recently such services have been described semantically in order to allow their automated (computer facilitated) discovery, invocation as well as inter-operation [19]. *Is it possible to integrate this technology with semantic wiki systems in order to support modern (dynamic) organizations?*

Another issue to consider is the issue of trust. One of the important layers of the semantic web stack is the trust layer. *How can one trust a semantic wiki application if there is no formal authority behind it that guarantees the trustworthiness of data?* Another type of contemporary self-organizing systems are social networking applications. Such applications allow individuals to connect through different mutual relations. By using social network analysis and especially by introducing a so called fishnet structure [37] one can "extract" the level of trustworthiness inside a social network.

[1] Compare to [60, 56]

In this research we shall take an object-oriented semantic modeling approach [58] an put our insights into a semantic wiki context. Most important objectives are that an autopoiesis facilitating semantic wiki system: (1) generates formalized knowledge that can be used for (computer-based) reasoning, (2) does not depend on end-user's knowledge about semantic technologies, and (3) evolves not only in terms of content but also in terms of functionality (as opposed to common wiki systems that evolve only in terms of content).[2]

We assume that the world being described on the system by its users is a set of objects that are in different mutual relations and interactions. Every object comprises eventually a set of relations with other objects, as well as a set of methods to be able to react on impulses (messages) from other objects. In a wiki context we shall call any wiki page an object. Thus we need to provide mechanisms to support the organized creation of such formalized objects.

We shall first take advantage of social tagging applications and introduce tags to wiki systems to provide meta data for wiki pages (objects). Hyperlinks shall represent the mutual relationships between objects. It should be possible to attach web services to any wiki page. These shall be the methods of the objects. In the end we take advantage of the social network of users surrounding a particular wiki. By analyzing the network we can provide trust levels for each and every meta data provided in the system.

To formalize this approach we need to formalize three things: (1) wiki languages (sometime also called wiki text, wiki syntax or markup language), (2) semantic wiki languages as well as (3) autopoiesis facilitating wiki languages. To do so we need to provide suitable formal tools like regular expressions (to formalize wiki languages), frame logic [41] (to formalize semantic wiki languages) as well as principles of annotation (to provide the trust levels) as well as principles of amalgamation (to provide a mechanism to combine formalized data) [50].

The semantic technologies shall be "hidden" to the end user. Users shall be able to normally use an autopoiesis facilitating semantic wiki systems as it were a "traditional" wiki system. By organizing their own knowledge through tags they unconsciously provide meta data for the semantic wiki system. By attaching web services to wiki pages they shall provide additional functionality to the system. Such functionality could provide a

[2]One should explicitly state here that the aim is not try to implement a system that should *achieve* autopoiesis. In fact, that would be quite difficult for an application system considered to be allopoietic. The main goal is to implement a system that will take advantage of various mechanisms in order to facilitate the autopoiesis of a social system that couples structurally to it.

suitable tool for dynamic information system integration.

1.1 Objectives and Hypotheses

The main aim of this thesis is to draw attention to the theory and practice of autopoiesis in the information and organization sciences with a special accent on different approaches to semantic wiki systems. We want to point out that social systems surrounding wiki systems are in their very nature autopoietic. This autopoiesis is facilitated through the ease of use and simplicity of wiki systems. By introducing complex semantic technologies to wiki systems this simplicity is lost. By "hiding" the semantic technologies into the background of the system as well as by introducing other autopoiesis facilitating technologies like social tagging and social networks we believe that this pitfall can be solved. A secondary goal is to provide a framework for knowledge management systems in modern organizations.

In order to establish a suitable formal backdrop for such systems we shall formalize the needed languages through well founded formalisms like regular expressions, frame logic and social network analysis.

The hypotheses of this thesis are as follows:

HYPOTHESIS 1 Through a formalization of wiki languages and semantic wiki languages as well as through the introduction of a social system's fishnet structure it is possible to establish a probability annotation scheme into semantic wiki languages.

HYPOTHESIS 2 Using the probability annotation of semantic wiki languages and by introducing social network analysis it is possible to establish an amalgamation scheme for such languages.

HYPOTHESIS 3 Using the probability annotation scheme as well as the amalgamation scheme a new language for autopoiesis facilitating semantic wiki systems can be established. The syntax and semantics of this language shall be formalized building upon the annotation and amalgamation schemes and the formalization of semantic wiki languages.

The scientific value of this thesis resides upon the explicit formalization of wiki languages, semantic wiki languages, the introduction of semantic web services to semantic wiki systems, and the introduction of an overall object-oriented approach into the formalization of semantic wiki languages. The main value is the establishment of a new language for autopoiesis facilitating semantic wiki systems through the introduction of

concepts borrowed from social tagging and social network analysis: attribute - value tags, probability (acquired through a special type of centrality) annotation, and amalgamation (acquired through social network integration). In the end a concrete implementation of such a language will be presented - the niKlas language.

On the other hand a critical review of autopoietic theory will be provided with special respect to its application to information and organizational sciences. It shall be shown that information systems are subsystems of organizations and social systems, and are in fact autopoietic. This conclusion provides a completely new research area in the field of information systems and IS/IT alignment.

From a social perspective a whole new field for new types of applications will be opened: autopoiesis facilitating applications. The very system that will be implemented in this thesis will support such new approaches and will be implemented using the open source approach and put into the public domain. Other values include the implementation of few example applications that shall be publicly available.

1.2 Methodology and Framework

Methodology is a scientific discipline whose main subject of study are the methods of scientific cognition. A framework is a way of purposeful problem solving. After a problem definition, stated objectives and hypotheses of scientific work, one needs to provide a set of suitable scientific methods that comprise the framework. A framework is also known to be the main idea and scope of systematic scientific work. It has to include a research plan that gives certain steps or phases of research [104].

In the context of this thesis the following methods will be used:

Parallel analysis and description. In a few sections we shall describe and analyze existing technologies and languages like existing wiki systems, semantic modeling approaches, ontologies, Semantic Web technologies and languages, semantic wiki systems, the concept of autopoiesis in information sciences, social networks as well as the fishnet organization.

Formalization of languages through regular expressions. A formalization of wiki languages through regular expressions will be provided. We shall introduce these languages alphabet and a set of regular expressions that will be able to match any word from such languages including hyperlinks, images, other objects, headings

and text formatting, comments, tables, variables, templates as well as references or citations.

Formalization in frame logic. A formalization of languages for semantic wiki systems, semantic web services, annotation and amalgamation scheme, and languages for autopoiesis facilitating semantic wiki systems shall be provided using the syntax of frame logic. We shall formalize the notions of class (type, concept), objects (instances), relations, attributes and methods provided through semantic web services. A set of rules for annotation as well as amalgamation shall also be provided.

Formalization of languages through EBNF grammar. A formalization of wiki languages, semantic wiki languages, probability annotated semantic wiki languages as well as amalgamated probability annotated semantic wiki languages through extended Backus-Naur notation will be provided. We shall introduce these languages alphabet and a set of regular expressions that will be able to match any word from such languages including hyperlinks, images, other objects, headings and text formatting, comments, tables, variables, templates, references, citations, queries and meta information.

Social Network Analysis. In order to provide a suitable framework for extracting probability from an autopoietic social system we shall use social network analysis. Particularly we shall use a special centrality measure (eigenvector centrality) in order to find actors (nodes) probabilities to state the right thing and to resemble a fishnet structure.

Database implementation. The very system (an autopoiesis facilitating semantic wiki system - ᴛᴀOPīs) will be implemented in an object -relational database using relational algebra and structured query language (SQL) and procedural languages (PL/pgSQL) for the PostgreSQL database management system.

Implementation in scripting languages. since the very system is a web application one needs to include scripting languages to add functionality. Python, an object-oriented scripting language will be used for this aim, and especially PL/PythonU its PotgreSQL version. Particularly the regular expression module (re) will be used for parsing, network connectivity modules for connecting hypertext transfer protocol (HTTP), post office protocol (POP) and simple mail transfer protocol (SMTP)

functionality, and operations and thread management modules to provide an interface to the $\mathcal{F}$LORA-*2* reasoning engine and other smaller tasks. On the other hand we shall use PHP (the hypertext preprocessor) scripting language mainly as a presentation layer using hypertext markup language (HTML), cascading style sheets (CSS) as well as JavaScript for additional functionality.

Implementation in frame logic. For the semantic web oriented part of the very system we will use the frame logic based language $\mathcal{F}$LORA-*2* . $\mathcal{F}$LORA-*2* is an object - oriented language for knowledge base, ontology an semantic web applications. We shall use $\mathcal{F}$LORA-*2* to implement a basic ontology that should be easily extended by users interaction.

The research plan is given in the following outline. In the chapter 1 the problem and subject of research definitions shall be provided. Afterwards objectives, scope and hypotheses as well as methodology and framework shall be defined. In chapter 2 we shall analyze wiki systems especially their history and important concepts. In chapter 3 we will formalize wiki languages using regular expressions. We will define their alphabet as well as regular expressions needed to match any word from wiki languages. In chapter 4 we will give a brief introduction to common semantic modeling approaches as well as technologies taking advantage of them. The following chapter 5 is concerned with the formalization of semantic wiki languages by introducing social tagging and frame logic. Chapter 6 gives an in-depth discussion of autopoietic theory as well as its possible application areas in the information sciences. Chapter 7 aims on formalizing a new language for autopoiesis facilitating semantic wiki systems by introducing social network analysis as well as annotation and amalgamation. Chapter 8 gives an outline of the **niKlas** language which is an implementation of such a language. Chapter 9 shows a few examples of projects on the TAOPIS system implemented during this research. In the end chapter 10 gives the final conclusions of this research as well as an evaluation of objectives accomplishment.

1.3 Related Work

There has been a fair deal of publishing presenting prototypes of semantic wiki systems [18, 45, 71, 81, 84, 87, 103, 101, 97, 24, 110, 38, 81, 42, 67, 4, 95, 34, 48, 16, 47, 71, 46, 3, 82, 29, 73, 79, 31] most of the building upon description logic, tagging (or structured tagging like in [92]) and link annotation. Some others introduce concepts

from cognitive psychology [52], intelligent agents to foster consistency [39], automated triplets from terms using thesaurus [36], contextual elicitation components [40], or special search facilities with keywords which are translated into structured, conjunctive queries [33]. Other authors introduced wiki generators which generate wiki systems based on some existing ontology [25], provided semantic wikis with formal tools to facilitate collaborative ontology development [43] or even developed application programming environments for the implementation of semantic wiki systems [90, 83]. Semantic wiki systems have been proven to be useful in lots of different fields including business [35], mathematics [47], biology [34] as well as the medical sciences [48]. Still there hasn't been any efforts to introduce weather semantic web services nor social network analysis. This thesis seems to be the first to introduce frame logic and an object - oriented approach to semantic wiki systems.

On the other hand there has been lots of publishing in the field of autopoietic theory but only few attempted to introduce autopoiesis in to the field of information sciences [100, 2, 96, 66]. Still this seems to be the first thesis that gives an elaborate conceptualization of autopoiesis in information systems and establishes its connections to other autopoietic systems like social systems and organizations.

Chapter 2

Wiki Systems

2.1 A Short History of Wiki's

The WikiWikiWeb was the first web site to be called a wiki [30, 115]. Ward Cunningham started the development of this system in 1994, and established a website at the c2.com domain on March 25th 1995. The term wiki originates also from Cunningham who remembered an employee on the Honolulu International Airport who recommended him to take the "Wiki Wiki" bus that travels between the airports terminals. Wiki means quick in Hawaiian what was the initial reason to use the word [22, 30, 115].

A partial inspiration for Cunningham came from Apple's HyperCard system that allowed its users to create virtual card stacks that one could interconnect. In a way, Cunningham further developed Vannevar Bush's idea of allowing users to comment and change their text mutually [23, 115].

In the early 2000s wiki systems are more frequently used by various organization for the collaboration of their employees in the context of communication during some project, intranet systems or documentation creation. Today it is more often the case that organizations use wiki systems as a substitute for intranet. Schools and faculties often use it to facilitate group learning which adds to the presumption that wiki usage is much broader than one would assume when investigating the public Internet [115].

Until 2001 wikis were a relatively unknown type of system, except in computer programmer's circles that used them more intensively. At that time the world became to know wikis, especially through the extreme success of Wikipedia, the free on line encyclopedia, that allowed anyone to edit content and articles [115].

Wikipedia was initially designed to be a supplement to Nupedia, also a free online encyclopedia that was started by Jimmy Wales. Nupidia included only articles written

by highly qualified experts. Articles were subject to a well founded reviewing process. Such an approach showed to be extremely slow and only 12 articles were completed during the first year of work, even if there were a relatively large number of interested editors on the mailing list, as well as an editor in chief (Larry Sager) who was employed by Wales. Wales and sager, after learning about the wiki concept, decided to enrich Nupedia with a wiki system. This system had to be only a supplement to Nupedia in order to allow easier publishing of (revised) articles [115].

To create a distinction the wiki system was launched at its own domain (wikipedia.com) on January 15th 2001. Initially the UseModWiki engine was used that was later replaced by a PHP based engine in January 2001, and finally with MediaWiki in July [115].

Wikipedia, after mentioned on well known sites like Slashdot and Kuro5in, gained a large number of associates and replaced Nupedia very quickly. In the first year over 20 000 articles were published with a constant growth from project launch. In mid 2009 it has over 2.8 million articles in its English version, whilst there are versions for almost any world language, with millions of editors around the world. On March 15th 2007 the word wiki entered the Oxford English Dictionary (on-line version) which shows how the term became very common [26, 115].

2.2 Important Concepts

A wiki system is heterarchic in nature. The concept of a wiki system resides on the following principles: every user or visitor of a wiki service is able to change existing content, to add new content and to discuss about it. Another mechanism which is built-into such systems is the possibility of interconnection of terms. Every term if mentioned in some article can be connected (hyperlinked) to other articles which elaborate it further. This mechanism gives users the possibility to find and understand unknown terms easier [88]. It could be said that communication is directed to achieve a purpose, in particular to create knowledge. Thus the advantages of wiki systems include goal attendance, direction of communication, and interconnection of terms.

Ward Cunningham, and co-author Bo Leuf, in their book The Wiki Way: Quick Collaboration on the Web [49] argued the essence of wiki systems as follows:

- *A wiki invites all users to edit any page or to create new pages within the wiki Web site, using only a plain-vanilla Web browser without any extra add-ons.*

- *Wiki promotes meaningful topic associations between different pages by making page link creation almost intuitively easy and showing whether an intended target page exists or not.*
- *A wiki is not a carefully crafted site for casual visitors. Instead, it seeks to involve the visitor in an ongoing process of creation and collaboration that constantly changes the Web site landscape.*

A minimalistic wiki system implementation would include two things (1) a mechanism for users to add/change/remove content and (2) a mechanism for users to interconnect content. The creation of new articles (or pages) is often achieved through the creation of an "non-existing page" link. In order to create a new page, a user creates link to it. When following the new link, an editor comes up that allows the user to create the new page. The page is off course connected through the initial link.

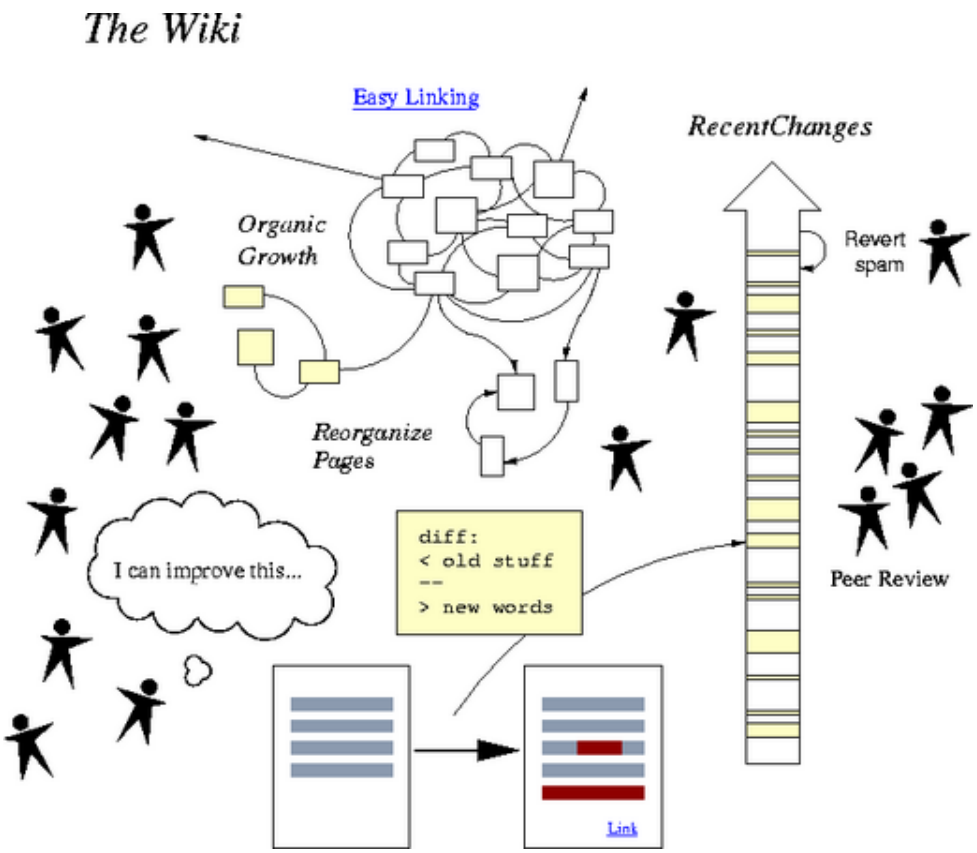

Figure 2.1: Important concepts in wiki systems [70]

More often, due to the evolution of the Web, wiki systems include additional mechanisms for adding non-textual content (images, multimedia, file attachments etc.), formatting content (text formatting, headers, tables, templates etc.), citation of relevant literature as well as other tools which ease their usage (rich text editors, visualizations etc.).

An important part of each wiki system is its syntax. Most wiki engines have their own syntax for content formatting and creation of hyperlinks. Some wiki systems allow their users to use rich text editors (so virtually no syntax is required). The syntax is then parsed and translated into HTML or some other representation language.

Most wiki systems include revision systems that allow users to track changes to a given article, see the history of it or to revert a page to some historic version for instance if some malicious user posted SPAM on it. The simple reorganization of pages is also often included into wiki systems. Today there are lots of wiki system implementations in almost any programming language [17].

Chapter 3

Languages for Wiki Systems

In the following we will concentrate on wiki syntax (often also called wiki text or wiki markup language). The notion of wiki language is in this context considered to be a synonym for the language used to format wiki articles, rather then the language used to implement a wiki system.

In order to provide a framework for describing the syntax of wiki systems, regular expressions [27], grammars and abstract syntax trees will be used [93]. Essential to the definition of regular expressions is the notion of language which provides a means of communication by sound and written symbols.

3.1 Language

A language definition consists basically of three parts [93]:

Syntax defines the ways in which symbols can be combined in order to create well formed sentences (or programs) in the language. Syntax provides a structural description of various valid strings in the language by defining the formal relations between the constituents of the language. Syntax is separated from meaning dealing only with the structure of the language.

Semantics defines the meaning of legal expressions in a language. In programming languages, semantics describe the behavior of the computer while executing a program dealing with input and output or steps that should be followed to execute the program on an abstract or concrete machine.

Pragmatics includes psychological and sociological aspects of a language such as utility, scope of application or effects on the user. For programming languages they

deal with issues like programming methodology, application efficiency or ease of implementation.

In the following we shall describe the syntax of wiki languages formally whilst the semantics and pragmatics will be described less formally in natural human language.

Definition An *alphabet* Σ is a finite set of letters.

Definition A *word* from alphabet Σ is a finite array of 0 or more letters from alphabet Σ.

A word with 0 letters is denoted with ε and is referred to as *empty word*.

Let Σ be an alphabet. Then Σ^n, where $n \geqslant 0$ denotes the set of all words of alphabet Σ that are of length n. Thus the set of all words over alphabet Σ is defined as:

$$\Sigma^* = \bigcup_{n \geqslant 0} \Sigma^n$$

Similarly the set of all non-empty words over alphabet Σ is defined as:

$$\Sigma^+ = \bigcup_{n \geqslant 1} \Sigma^n$$

Sometimes its convenient to observe words as discrete functions. Every word $w \in \Sigma^*$ can be defined as $w : \{1, ..., |w|\} \to \Sigma$, where $w(i)$ is the i-th letter of word w.

We say that letter σ has an *occurrence* i word w if there exists a $j \in \{1, ..., |w|\}$ such that $w(j) = \sigma$.

Definition Let Σ be an alphabet and let $v, w \in \Sigma^*$ be words. The operation of *concatenation* between v and w is then defined as word $z = vw$ provided that:

- $z(i) = v(i)$ for $i \in \{1, ..., |v|\}$, and
- $z(j + |v|) = w(j)$ for $j \in \{1, ..., |w|\}$

It is clear that for every word $w \in \Sigma^*$ $w\sigma = w$. It is also clear that concatenation is an associative operation, e.g. $u(vw) = (uv)w = uvw$ for any words $u, v, w \in \Sigma^*$.

Definition Let Σ be an alphabet and $v, w \in \Sigma^*$ be words.We say that word v is a *sub-word* of word w if there are words $t, u \in \Sigma^*$ such that $w = tvu$.

If $t = \sigma$, e.g. if $w = vu$, then word v is called a *prefix* of word w

If $u = \sigma$, e.g. if $w = tv$, then word v is called a *suffix* of word w

Given the word $w \in \Sigma^*$ over alphabet Σ, we define further:

- $w^0 = \sigma$
- $w^k = w^{k-1}w$.

Having the basic terms defined we can now define language:

Definition Language $\mathcal{L} \subseteq \Sigma^*$ over alphabet Σ is any set of words from Σ^*.

3.2 Regular Expressions

Regular expressions are an important way to represent languages, but besides in computer theory, they play a major role in real world applications [27]. They are often used in text searches using special characters, so called *wildcards*. *NIX operating systems, for instance, provide full support for regular expressions as well as do most programming languages.

Definition Let Σ be an alphabet. Then the following are *regular expressions* over Σ:

- $\emptyset$ is a regular expressions.
- If $\sigma \in \Sigma$, then σ is a regular expression.
- If v and w are regular expressions, then vw is also a regular expression.
- If w is a regular expression, then $(w)*$ is a regular expression.
- If v and w are regular expressions, then $(v)|(w)$ is a regular expression.

Herewith we have defined the syntax of regular expressions.

It must be noted that every regular expression over some alphabet Σ defines a language over Σ. For example if $\Sigma = \{x, y, z\}$ then the following regular expressions over Σ would each define a language:

- $r_1 = (z)*$
- $r_2 = (y)|(z)$
- $r_3 = ((zx)*)|((y)*)$

Some of the parenthesis can be omitted:

- $r_1 = z*$
- $r_2 = y|z$
- $r_3 = (zx) * |(y*)$

The semantics of regular expressions are of special interest. Regular expressions represent templates for constructing words of a language. The special characters have simple meanings. The $*$ sign denotes 0 or more occurrences of the regular expression preceding it. This means that the place where an expression $(w)*$ occurs in a regular expression defining the words of the language, can eventually be a sub-word that is defined by regular expression w occurring 0 or more times. The | sign defines disjunction. The place in a regular expression where $(w)|(v)$ occurs, the word of the language defined will either contain a sub-word defined by w or a sub-word defined by v.

So the previously shown expressions would define languages consisting of:

- $\mathcal{L}_1 = \{z, zz, zzz, zzzz, ...\}$
- $\mathcal{L}_2 = \{y, z\}$
- $\mathcal{L}_2 = \{zx, zxzx, zxzxzx, ..., y, yy, yyy, ...\}$

It is sometimes convenient to define additional special characters for special tasks. In the following we will use the following set of equations.

$$w+ = ww*$$

Where w is any regular expression defined over Σ. $+$ denotes one or more occurrences.

$$w^n = \overbrace{ww...w}^{n\times}$$

The regular expression w is repeated n times (power).

$$(\sigma_1, ..., \sigma_n)! = \varsigma_1|...|\varsigma_m$$

Where $\varsigma_1, ..., \varsigma_m, \sigma_1, ..., \sigma_n \in \Sigma$ and $\{\varsigma_1, ..., \varsigma_m\} = \Sigma - \{\sigma_1, ..., \sigma_n\}$. ! denotes letter set negation, e.g. the regular expression will match any letter not in the defined set. This is especially convenient with large alphabets.

$$w! = \neg w$$

Where w is a regular expression defined over Σ. $w!$ will match a word if and only if w does not match it.

$$. = \sigma_1|...|\sigma_n$$

Where $\{\sigma_1, ..., \sigma_n\} = \Sigma$. The . defines any letter from Σ.

$$w? = (w^0)|(w^1)$$

Where w is a regular expression defined over Σ. ? defines optionality.

Definition Let v and w be to regular expressions over alphabet Σ. $v \Subset w$ denotes that all words defined by w are also defined by v. The opposite does not necessarily hold.

3.3 Grammars

Grammars are a formal method for describing the syntax of languages [93].

Definition A grammar $\langle \Sigma, N, P, S \rangle$ consists of four parts:

1. A finite set Σ of **terminal symbols**, the **alphabet** of the language, that are assembled to make up the sentences in the language.
2. A finite set N of **non-terminal symbols** or **syntactic categories**, each of which represents some collection of subphrases of the sentences.
3. A finite set P of **productions** or **rules** that describe how each non-terminal is defined in terms of terminal symbols and non-terminals. The choice of nonterminals determines the phrases of the language to which we ascribe meaning.
4. A distinguished non-terminal S, the **start symbol**, that specifies the principal category being defined - for example sentence or program.

In the following we will use a metalanguage called Backus-Naur form or BNF to describe the grammars of wiki languages, semantic wiki languages as well as autopoiesis facilitating semantic wiki languages. Nonterminals in BNF have the form ⟨category-name⟩ and production rules are written as for example:

$$\langle \text{declaration} \rangle \ ::= \ [\text{int}] \ \langle \text{variable name} \rangle \ [=] \ \langle \text{expression} \rangle \ [;]$$

In this example expression enclosed in square brackets int, = and ; are terminal symbols of the language. The symbol ::= can be read as "is defined to be" or "can be composed of" and is part of BNF.

Definition The **vocabulary** of a grammar includes its terminal and non-terminal symbols. An arbitrary production has the form $\alpha ::= \beta$ where α and β are strings of symbols from the vocabulary, and α has at least one non-terminal in it.

According to Chomsky grammars can be classified into four categories (type 0 to type 3) depending on their structure.

Type 0 are the most general or **unrestricted grammars**. They require only that at least one non-terminal occurs on the left hand side of a rule.

$$\alpha \ ::= \ \beta$$

Type 1 are **context-sensitive grammars** require additionally that the right hand side contains no fewer symbols than the left.

$$\alpha \ \langle \text{B} \rangle \ \gamma \ ::= \ \alpha \ \beta \ \gamma$$

where B is a non-terminal and α, β and γ are strings over the vocabulary whereby β in a non-empty string.

Type 2 represent the **context-free** grammars that prescribe that the left hand side of productions be a single non-terminal symbol.

$$\langle \text{A} \rangle \ ::= \ \alpha$$

Type 3 are the most restrictive grammars or **regular grammars** and allow only a terminal or a terminal followed by a non-terminal symbol on the right side.

$$\langle \text{A} \rangle \ ::= \ \alpha$$

or

$$\langle \mathrm{A} \rangle \ ::= \ \alpha \ \langle \mathrm{A} \rangle$$

The **extended Backus Naur form** or EBNF defines additional syntactic conventions in BNF. In particular these include the $*$ operator (an expression may be repeated zero or more times), the + operator (an expression may be repeated one or more times), the ? operator (an expression is optional), the | operator (representing a possible choice between two expressions) as well as parentheses (and) that allow the grouping of expressions. These additions are similar to the regular expressions defined above.

3.4 Wiki Syntax

In the following we will use regular expressions and EBNF to define the language of wiki systems. We use regular expressions since they almost allow for a direct language parser implementation and use EBNF to provide a higher abstraction of syntax. We will presume that the alphabet of wiki languages Σ_W consists of UNICODE characters but the following formalization should apply to any set of human language characters. Since UNICODE includes the special characters we shall use the “
” character for escaping them. Also common shorthands for UNICODE characters are A-Z (all uppercase characters), a-z (all lowercase characters) and 0-9 (numeral characters).

The grammar of wiki languages is given in the following listing:[1]

```
<wiki_page> ::= <statement>*
<statement> ::= <STRING>
              | <formatting_expression>
              | <display_object>
              | <comment>
              | <hyperlink>
              | <table>
              | <variable_template>
              | <reference_entry>
              | <reference_citation>
```

[1]For a complete XSB Prolog implementation of a niKlas plain wiki syntax parser please refer to appendix A.

Whereby `<STRING>` represents a non-terminal defining any string that does not conflict with the concrete wiki language keywords and `<URL>` represents a non-terminal defining any URL. As we can see the start symbol is defined as `<wiki_page>` which can be any number of statements (including an empty statement). The vocabulary of a wiki language thus includes strings, formatting expressions, display objects (including images), comments, hyperlinks (internal and external), tables, variables, templates, reference entries as well as reference citations.

3.4.1 Hyperlinks

Hyperlinks are an important feature of wiki languages. There are two types of hyperlinks: internal and external. Internal hyperlinks point to pages inside the wiki system, while external point to external URLs. A basic production rule in EBNF for hyperlinks is given in the following listing:

```
<hyperlink> ::=
  <hyperlink_start>
  <URL>
  <hyperlink_name>
  <hyperlink_end>
```

Definition Let l_{begin} be a regular expression that matches a link word beginning, l_{int} a regular expression that matches all possible internal wiki URLs, l_{ext} a regular expression that matches all external URLs, $l_{\text{delimiter}}$ a regular expression that matches a delimiter, l_{name} a regular expression matching the link's name, and l_{end} a regular expression that matches a link word end, whereby all are of them defined over Σ_W. Let the following relations hold:

$$
\begin{aligned}
l_{\text{int}} &\notin l_{\text{link delimiter}} \\
l_{\text{ext}} &\notin l_{\text{link delimiter}} \\
l_{\text{delimiter}} &\notin l_{\text{end}} \\
l_{\text{name}} &\notin l_{\text{end}}
\end{aligned}
$$

Hyperlinks are then defined with the regular expression:

$$r_{\text{hyperlink}} = l_{\text{begin}}(l_{\text{int}}|l_{\text{ext}})(l_{\text{delimiter}}l_{\text{name}})?l_{\text{end}}$$

For example if we take the following regular expressions (**niKlas** syntax):

$$\begin{aligned} l_{\text{begin}} &= [\text{link}= \\ l_{\text{int}} &= (>)!* \\ l_{\text{ext}} &= (>)!* \\ l_{\text{delimiter}} &= > \\ l_{\text{name}} &= (])!* \\ l_{\text{end}} &=] \end{aligned}$$

Than the following would be instances of hyperlinks:

```
[link=wikipage>link to another wiki page]
[link=http://www.foi.hr>link to FOI]
```

If we would have used the wiki syntax parser from appendix A to parse the above hyperlinks we would obtain the following parse tree:

```
wiki_page(
 statements(
  [
   statement(
    hyperlink(
     hyperlink_start(
      [link=
     ),
     internal_url(
      wikipage
     ),
     link_name(
      link_to_another_wiki_page
     ),
```

```
      hyperlink_end(
       ]
      )
     )
    ),
    statement(
     hyperlink(
      hyperlink_start(
       [link=
      ),
      external_url(
       http://www.foi.hr
      ),
      link_name(
       link_to_FOI
      ),
      hyperlink_end(
       ]
      )
     )
    )
   ]
  )
)
```

Or to simulate the MediaWiki syntax[2] partially[3]:

[2]The MediaWiki syntax is used by Wikipedia for instance.

[3]The MediaWiki syntax has also special <nowiki> tags that allow the escape of wiki syntax

$$
\begin{aligned}
l_{\text{begin}} &= [\\
l_{\text{int}} &= (\backslash|)! * \\
l_{\text{ext}} &= (\text{A-Za-z0-9.\_}\backslash\backslash\tilde{}\%\text{-+\&\#?!=}\backslash(\backslash)@)! * \\
l_{\text{delimiter}} &= (\backslash|)|(\ ) \\
l_{\text{name}} &= (])! * \\
l_{\text{end}} &=]
\end{aligned}
$$

Than the following would be instances of hyperlinks:

```
[wikipage|link to another wiki page]
[http://www.foi.hr link to FOI]
[http://www.news.at]
```

3.4.2 Images and other Objects

Images and other object including video and audio material, attachments and/or plug-in dependable objects (like shockwave flash objects etc.) are an essential part of wiki systems. A basic production rule in EBNF for images and other display objects is given in the following listing:

```
<display_object> ::=
  <display_object_start>
  <URL>
  <display_object_options>?
  <display_object_end>

<display_object_options> ::=
  <display_object_option>+
```

Definition Let i_{begin} be a regular expression that matches an image or other object's word beginning, i_{object} a regular expression matching all possible image or object's words, $i_{\text{delimiter}}$ a regular expression that matches a delimiter, i_{options} a regular expression that

matches all possible options for an image or object, and i_{end} a regular expression that matches an image or other object's word ending, whereby all of them are defined over Σ_W. Let the following set of relations hold:

$$\begin{aligned} i_{\text{object}} &\notin i_{\text{delimiter}} \\ i_{\text{options}} &\notin i_{\text{end}} \end{aligned}$$

Then *images and other objects* are defined with the following regular expression:

$$r_{\text{object}} = i_{\text{begin}} i_{\text{object}} (i_{\text{delimiter}} i_{\text{options}})? i_{\text{end}}$$

If take, for instance, the following regular expressions (**niKlas** syntax for images):

$$\begin{aligned} i_{\text{begin}} &= [\text{img}= \\ i_{\text{object}} &= (\ )!* \\ i_{\text{delimiter}} &= (\ ) \\ i_{\text{options}} &= ((\text{width}=(0\text{-}9)*\%?)?(\ )?(\text{height}=(0\text{-}9)*\%?)?) \\ i_{\text{end}} &=] \end{aligned}$$

Then the following would be instances of images:

```
[img=http://www.foi.hr/image.jpg]
[img=http://www.google.com/image2.jpg width=20%]
[img=http://www.news.at/image3.gif width=100 height=20]
```

If we parsed the first image using the parser defined in appendix A we would obtain the following derivation tree.

```
wiki_page(
 statements(
  [
```

```
    statement(
     image(
      image_start(
       [img=
      ),
      external_url(
       http://www.foi.hr/image.jpg
      ),
      image_end(
       ]
      )
     )
    )
   ]
 )
)
```

Or to simulate the WikiMedia syntax:

$$
\begin{aligned}
i_{\text{begin}} &= [[\text{Image:} \\
i_{\text{object}} &= (\backslash|)! * \\
i_{\text{delimiter}} &= (\backslash|) \\
i_{\text{options}} &= (])!* \\
i_{\text{end}} &=]]
\end{aligned}
$$

Than the following would be instances of images:

```
[[Image:wiki.png]]
[[Image:wiki.png|Wikipedia, The Free Encyclopedia.]]
[[Image:wiki.png|30 px]]
[[Image:wiki.png|right|Wikipedia Encyclopedia]]
```

3.4.3 Headings and Text Formatting

Content structuring and text formatting is another important feature. There are lots of different formatting rules, but they can mostly be described through a simple regular expression. A basic production rule in EBNF for headings and text formatting expressions is given in the following listing:

```
<formatting_expression> ::=
  <formatting_expression_start>
  <statement>
  <formatting_expression_end>
```

Definition Let t_{begin} be a regular expression that matches a formattings word beginning, t_{text} a regular expression matching all possible words of text to be formatted, and t_{end} a regular expression that matches a formattings word ending, whereby all of them are defined over Σ_W. Let the following relation hold:

$$t_{\text{text}} \notin t_{\text{end}}$$

Then a *formatting* is defined with the following regular expression:

$$r_{\text{text}} = t_{\text{begin}} t_{\text{text}} t_{\text{end}}$$

As an example we will take the heading formatting of the **niKlas** syntax defined with regular expressions similar to the following:

$$\begin{aligned} t_{\text{begin}} &= [\text{h1}] \\ t_{\text{text}} &= ([\text{h1}])!^* \\ t_{\text{end}} &= [/\text{h1}] \end{aligned}$$

The following is an instance of headings:

```
[h1]Big heading[/h1]
```

If we parsed this expression with the **niKlas** parser from appendix A, the following derivation tree would be obtained.

```
wiki_page(
 statements(
  [
   statement(
    formatting_expression(
     formatting_expression_start(
      [h1]
     ),
     statements(
      [
       text(
        Big_heading
       )
      ]
     ),
     formatting_expression_end(
      [/h1]
     )
    )
   )
  ]
 )
)
```

Or to simulate WikiMedia syntax for first level headings (where \n defines a newline character):

$$
\begin{aligned}
t_{\text{begin}} &= \backslash\text{n}== \\
t_{\text{text}} &= (==\backslash\text{n})!* \\
t_{\text{end}} &= ==\backslash\text{n}
\end{aligned}
$$

The following is then an instance of headings:

```
== Introduction ==
```

3.4.4 Comments

Some wiki languages provide commenting facilities. Their formalization is essentially equivalent to text formattings. A basic production rule in EBNF for comments is given in the following listing:

```
<comment> ::=
  <comment_start>
  <statement>
  <comment_end>
```

Definition Let c_{begin} be a regular expression that matches a commentary word beginning, c_{text} a regular expression matching all possible words of commentary text, and c_{end} a regular expression that matches a commentary word ending, whereby all of them are defined over Σ_W. Let the following relation hold:

$$c_{\text{text}} \notin c_{\text{end}}$$

Then a *comment* is defined with the following regular expression:

$$r_{\text{comment}} = c_{\text{begin}} c_{\text{text}} c_{\text{end}}$$

As an example we take the WikiDot syntax :

$$\begin{aligned} c_{\text{begin}} &= [\backslash!\text{--} \\ c_{\text{text}} &= (\text{--}])!^* \\ c_{\text{end}} &= \text{--}] \end{aligned}$$

The following is an instance of a comment:

```
[!-- invisible comment --]
```

3.4.5 Tables

A basic set of production rules in EBNF for tables is given in the following listing:

```
<table> ::=
  <table_start>
  <table_row>*
  <table_end>

<table_row> ::=
  <table_row_start>
  <table_cell>*
  <table_row_end>

<table_cell> ::=
  <table_cell_start>
  <statement>*
  <table_cell_end>
```

Definition Let tb_{begin} be a regular expression that matches a table word beginning, tb_{text} a regular expression matching all possible words inside of tables, $tb_{\text{cell delimiter}}$ a regular expression that matches a table cell delimiter word, $tb_{\text{row delimiter}}$ a regular expression that matches a table row delimiter word, and c_{end} a regular expression that matches a table word ending, whereby all of them are defined over Σ_W. Let the following set of relations hold:

$$
\begin{aligned}
tb_{\text{text}} &\notin tb_{\text{cell delimiter}} \\
tb_{\text{text}} &\notin tb_{\text{row delimiter}} \\
tb_{\text{text}} &\notin tb_{\text{end}} \\
tb_{\text{cell delimiter}} &\notin tb_{\text{row delimiter}} \\
tb_{\text{row delimiter}} &\notin tb_{\text{cell delimiter}} \\
tb_{\text{cell delimiter}} &\notin tb_{\text{end}} \\
tb_{\text{row delimiter}} &\notin tb_{\text{end}}
\end{aligned}
$$

Then a *table* of size $n \times m$ is defined with the following regular expression:

$$r_{\text{table}} = tb_{\text{begin}}((tb_{\text{text}}tb_{\text{cell delimiter}})^n tb_{\text{row delimiter}})^m tb_{\text{end}}$$

For instance, the niKlas syntax uses regular expressions similar to the following to parse tables:

$$\begin{aligned} tb_{\text{begin}} &= \text{[table]} \\ tb_{\text{text}} &= ((\&\&)|(\#\#)|([/\text{table}]))!^* \\ tb_{\text{cell delimiter}} &= \&\& \\ tb_{\text{row delimiter}} &= \#\# \\ tb_{\text{end}} &= \text{[/table]} \end{aligned}$$

Thus the following would be an instance of a table:

```
[table]
row 1 column 1 && row 1 column 2 && row 1 column 3 ##
row 2 column 1 && row 2 column 2 && row 2 column 3 ##
row 3 column 1 && row 3 column 2 && row 3 column 3
[/table]
```

If we would parse this table with the parser from appendix A, we would obtain the following parse tree:

```
wiki_page(
 statements(
  [
   statement(
    wtable(
     wtable_start(
      [table]
     ),
     wtable_rows(
      [
```

```
      wtable_row(
       wtable_cells(
        [
         wtable_cell(
          statements(
           [
            text(
             row_1_column_1
            )
           ]
          )
         ),
         wtable_cell(
          statements(
           [
            text(
             row_1_column_2
            )
           ]
          )
         ),
         wtable_cell(
          statements(
           [
            text(
             row_1_column_3
            )
           ]
          )
         )
        ]
       )
      ),
      wtable_row(
```

```
      wtable_cells(
       [
        wtable_cell(
         statements(
          [
           text(
            row_2_column_1
           )
          ]
         )
        ),
        wtable_cell(
         statements(
          [
           text(
            row_2_column_2
           )
          ]
         )
        ),
        wtable_cell(
         statements(
          [
           text(
            row_2_column_3
           )
          ]
         )
        )
       ]
      )
     ),
     wtable_row(
      wtable_cells(
```

```
      [
       wtable_cell(
        statements(
         [
          text(
           row_3_column_1
          )
         ]
        )
       ),
       wtable_cell(
        statements(
         [
          text(
           row_3_column_2
          )
         ]
        )
       ),
       wtable_cell(
        statements(
         [
          text(
           row_3_column_3
          )
         ]
        )
       )
      ]
     )
    )
   ]
  ),
  wtable_end(
```

```
      [/table]
     )
    )
   )
  ]
 )
)
```

3.4.6 Variables and Templates

Some wiki languages provide variables (e.g. current page title, current date, current category etc.) and templates (e.g. stub, outline, table of content etc.). A basic production rule in EBNF for variables and templates is given in the following listing:

```
<variable_template> ::= <STRING>
```

Such words are easily defined using regular expressions:

Definition Let v_{template} be a regular expression that denotes all possible variable and template words defined over Σ_W. Then the following regular expression defines *variables and templates*:

$$r_{\text{template}} = v_{\text{template}}$$

3.4.7 References

References allow users to reference sources that were used to write the current wiki page. In essence there are to parts of a reference: (1) an entry - probably given at the and of the wiki page, and (2) one or more references to the listing. A basic set of production rules in EBNF for reference entries and reference citations is given in the following listing:

```
<reference_entry> ::=
  <reference_entry_start>
  <reference_cite_key>
  <statement reference_entry_end>

<reference_citation> ::=
```

```
<reference_citation_start>
<reference_cite_key>
<reference_citation_name>
<reference_citation_end>

<reference_cite_key> ::= <STRING>
```

Definition Let e_{begin} be a regular expression that matches references entry word's beginning, e_{name} a regular expression that matches all possible reference name words, e_{entry} a regular expression matching the actual entry word, and e_{end} a regular expression matching the references entry word's end, whereby all of them are defined over Σ_W. Let the following relation hold:

$$e_{\text{name}} \quad \notin \quad e_{\text{end}}$$

Then a *reference entry* is defined with the following regular expression:

$$r_{\text{entry}} = e_{\text{begin}} e_{\text{name}} e_{\text{end}} (e_{\text{entry}})?$$

Definition Let ref_{begin} be a regular expression that matches a reference word's beginning, e_{name} a regular expression that matches all possible reference name words that is defined inside an entry on the given wiki page under consideration, ref_{title} a regular expression matching all possible reference title words, $ref_{\text{delimiter}}$ a regular expression matching reference delimiter words, and ref_{end} a regular expression matching the references word's end, whereby all of them are defined over Σ_W. Let the following set of relations hold:

$$\begin{aligned} e_{\text{name}} \quad &\notin \quad ref_{\text{delimiter}} \\ ref_{\text{title}} \quad &\notin \quad ref_{\text{end}} \end{aligned}$$

Then a *reference* is defined with the following regular expression:

$$r_{\text{reference}} = ref_{\text{begin}} e_{\text{name}} ref_{\text{delimiter}} ref_{\text{title}} ref_{\text{end}}$$

For example the WikiDot syntax uses regular expressions similar to the following:

$$\begin{aligned}
e_{\text{begin}} &= : \\
e_{\text{name}} &= (\ ,:)!* \\
e_{\text{end}} &= \ : \\
e_{\text{entry}} &= .* \\
ref_{\text{begin}} &= [((\text{bibcite} \\
ref_{\text{delimiter}} &= \emptyset \\
ref_{\text{title}} &= \emptyset \\
ref_{\text{end}} &=))]
\end{aligned}$$

So the following would be an instance of a reference entry with corresponding reference:[4]

```
The first pulsar was observed by J. Bell and
A. Hewish [((bibcite bell))].

: bell : Bell, J.; Hewish, A.; Pilkington, J. D. H.;
Scott, P. F.; and Collins, R. A. //Observation of
a Rapidly Pulsating Radio Source.// Nature 217,
709, 1968.
```

The niKlas syntax is defined by the following regular expressions:

[4]The entry must occur inside a bibliography block.

$$
\begin{aligned}
e_{\text{begin}} &= [\text{ref}= \\
e_{\text{name}} &= (])!* \\
e_{\text{end}} &=] \\
e_{\text{entry}} &= .* \\
ref_{\text{begin}} &= [\text{cite}= \\
ref_{\text{delimiter}} &= (¿) \\
ref_{\text{title}} &= (])!* \\
ref_{\text{end}} &=]
\end{aligned}
$$

Thus the following is a valid entry with corresponding citation:

```
To use [cite=references2009>references] and citations
use the following syntax:

[ref=references2009] Luhmann, N. Soziale systeme, 1984.
```

If parsed with the **niKlas** parser from appendix A the following parse tree would be obtained:

```
wiki_page(
 statements(
  [
   text(
    To_use
   ),
   statement(
    reference_citation(
     reference_citation_start(
      [cite=
     ),
     cite_key(
      references2009
```

```
      ),
      reference_citation_name(
       references
      ),
      reference_citation_end(
       ]
      )
     )
    ),
    text(
     and_citations_use_the_following_syntax:
    ),
    statement(
     reference_entry(
      reference_entry_begin(
       [ref=
      ),
      cite_key(
       references2009
      ),
      reference_entry_end(
       ]
      )
     )
    )
   ]
  )
)
```

In the and we can now define the (basic) wiki language:

Definition A *wiki language* $\mathcal{L}_W$ is defined through the following regular expression defined over Σ_W:

$$r_W = (r_{\text{hyperlink}})|(r_{\text{object}})|(r_{\text{text}})|(r_{\text{comment}})|(r_{\text{table}})|(r_{\text{template}})|(r_{\text{entry}})|(r_{\text{reference}})$$

This basic language can be extended by adding additional regular expressions for additional features of a wiki language.

Chapter 4

From Semantic Modeling over the Semantic Web to Semantic Wiki Systems

By modeling a domain (be it using an ontology, knowledge base, UML[1] diagram or any other formalism) one expresses her own knowledge about the domain. This in particular means that the main concept in modeling is knowledge. I. Nonaka once stated that knowledge is personal, a "justified true belief" [69]. Thus one implicitly presumes that the data expressed in ones domain model is true. If we ask now *what is the truth* we come to one of the fundamental questions in philosophy. F. Nietzsche argued that one cannot prove the truth which is nothing more than the invention of fixed conventions for merely practical purposes, especially those of repose, security and consistence. According to this view, no one can prove that the author of this text or the whole world isn't just a fantasy of the reader reading this article.

Nonaka's definition includes, by intention or not, two more crucial words: *justified* and *belief*. An individual will consider something to be true that he believes in, and from that perspective, the overall truth will be a set of statements that the community believes in. This mutual belief makes this set of statements justified. The truth was once that the Earth was flat until philosophers and scientists started to question this theory. The Earth was also once the center of the universe. So an interesting fact about the truth is that it evolves depending on the different beliefs of a certain community.

In an environment where a community of individuals collaborates in modeling a domain there is a chance that there will be certain disagreements about the domain

[1] Unified Modeling Language

which yield certain inconsistencies in the model or ask for an overall consensus. A good example of such disagreements are the so called "editor wars" on Wikipedia the popular free on-line encyclopedia. A belief about the War in Iraq will most probably differ between an American and an Iraqi but they will most probably share the same beliefs about fundamental mathematical algebra.

The main question that should be asked is if it is possible to apply this concept of multiple views or beliefs about a domain (that we shall call aspects as follows) into systems for collaborative knowledge base development, as well as if such a conceptualization can yield a more realistic model of a given domain? In the following a brief description of important concepts is given in order to build a new framework.

4.1 Semantic Modeling and the Object - oriented Paradigm

Semantics are meaning and thus semantic modeling is the design of meaning. The object-oriented paradigm is a very popular framework in information and knowledge engineering, programming and simulation modeling these times. This approach is in a way natural, since the world around us is comprised of objects that are in some mutual interactions. For instance, at this very moment You are an object that is interacting with another object (this thesis).

Its fundamental to consider the term concept for any discussion about semantic modeling. The world of concepts around us can be observed statically by introducing terms like domains, objects, relations, attributes, generalizations etc. and dynamically by introducing states, events, methods, triggers, control conditions, state changes, aggregations, constraints, rules, meta models, power-types etc. For the sake of this thesis we shall concentrate on a restriction of these terms as argued further [2].

4.1.1 Domains

A domain or context is a collection of objects in a chosen field of interest. A domain specification is a collection of concepts which refer to the domain. For example if we take as a domain a classroom, then concepts inside it would be similar to student, chair, blackboard, teacher etc. The domain is the "sphere of activity and influence" since inside

[2]For an in-depth discussion on object-oriented semantic modeling please refer to [58]

it objects fulfill their activities and influence other objects.

4.1.2 Concepts

Concepts are constructs in the human mind that allow us to reduce the complexity of the world around us. If we wouldn't have concepts the world would for us be just a huge mess of objects that we wouldn't be able to recognize and distinct of each other. For example imagine how an ancient cavemen would had defined an airplane that flew over his head. He, off course didn't have an the necessary concept that would allow him to define an airplane. He would have probably been pretty amazed by the giant metal bird. Every person has a lots of concepts defined in its mind that were acquired during its life.

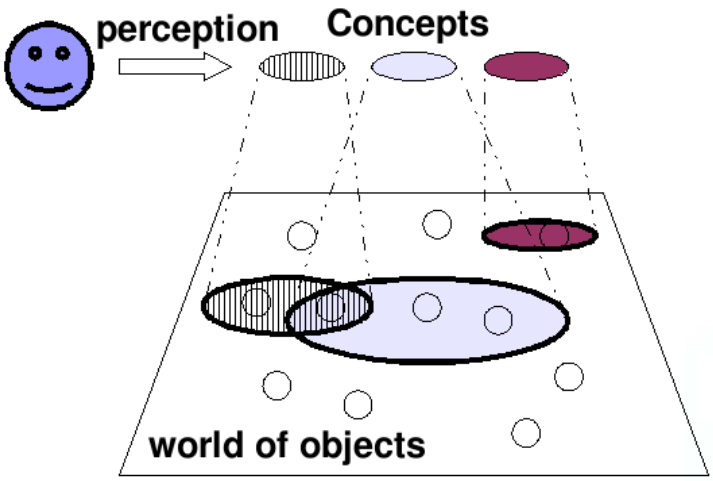

Figure 4.1: Concepts as filters in our perception [58]

As shown on figure 4.1, concepts are filters that allow one to filter the complex world of objects. According to J. Martin & J. Odell a concept is an idea or meaning that we assign to things in our mind [58]. People are able to create them, assign symbols to them and manipulate these symbols in order to determine and communicate meaning. Every concept can be defined with a triad, whereby the symbol is nonobligatory as shown on figure 4.2.

One could state that concepts are units of knowledge that consist of:

1. an **intention** - a concise definition of the concepts that includes a test that decides if the concept is applicable to some object or not
2. an **extension** - the set of all objects to which the concept is applicable
3. a **symbol** - a concise way to denote objects.

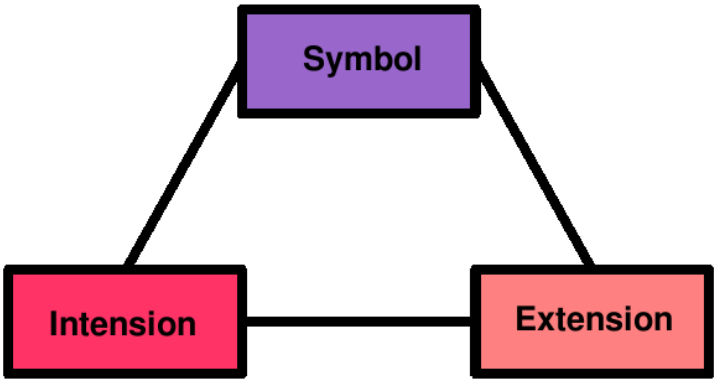

Figure 4.2: The concept triad [58]

As stated before, concepts can be used without symbols that denote them, but when talking about implementation there can be also partial definitions of concepts.[3] Symbols are mostly used for communication. The implementation of soncepts are often classes or types, especially in programming languages.

4.1.3 Generalization and Specialization

Generalization is the operation or result of identifying a concept (class, type) that fully includes or subsumes another concept. Generalization allows us to state that all instances of some specific type, are also instances of another, more general type. The opposite does not necessarily have to hold. For example, one could state that all sandals are footwear, but not all footwear are sandals since there are other types of footwear like shoes, boots, high-heals etc.

Specialization is the inverse operation or result of generalization as shown on figure 4.3

Generalized types are super-types (superclasses), whilst specialized types are subtypes (subclasses). Super-types have a more general intention then their subtypes. The intention of subtypes is particularly more specialized or more strict and rigorous. Generalization and specialization allow us to create hierarchies of concepts.

[3] A concept is partially defined if one component of the triad is missing. Thus we an have concepts without intension (we have a symbol - e.g. XY and a set of objects that the concept applies to e.g. $\{2, Q, \odot, ♁, ♡\}$, but we are unable to define the concept due to some reason), concepts without extension (we have a symbol - e.g. perfect student, and a defintion - e.g. a student with all straight A's and all extraordinary activities, degrees etc. but no student does comply to this definition), and concepts without symbol (we have a definition - e.g. a space object with properties XY and intension $\{o_1\}$ but no symbol yet).

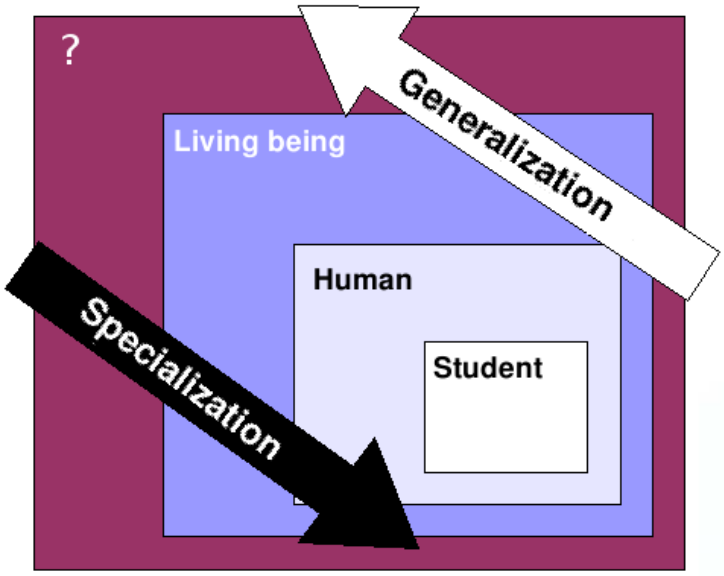

Figure 4.3: Generalization and specialization [58]

4.1.4 Objects

An object is anything a concept can be applied on. An object is an instance of some concept. Most objects have their life-cycles which in particular means that one can define the beginning and end of existence. For example, for object that are instances of concept *human being* their beginning of existence is their birth and their end is their death. To describe the changes of an object during its life-cycle we use sets.

When applying a concept to an object, we have classified the object as en element of a set, thus incrementing the sets cardinality by one. In the opposite case, when declassifying an object, the object is extracted from the set, and the cardinality is decremented by one.

Some objects are classified and declassified during their life-cycles. They are members of some set at one time, and after some process or event they aren't anymore. Such changes can happen as long as the object exists.

As one can see of figure 4.4 object Nick entered the set *Person* on its birth. After some time, Nick got into college and became additionally a member of the set defined by concept *Student*. After studying some time he graduated, and was thus declassified from the set of all students. Later on he managed to get a job, entering thereby the set defined by concept *Employee*. Since his boss was generous she allowed him a grant to take a postgraduate study, which made Nick again a member of the set of all students, being an employee at the same time. After graduating he again left the student set.

Any object can have multiple concepts applicable to it. This phenomenon is

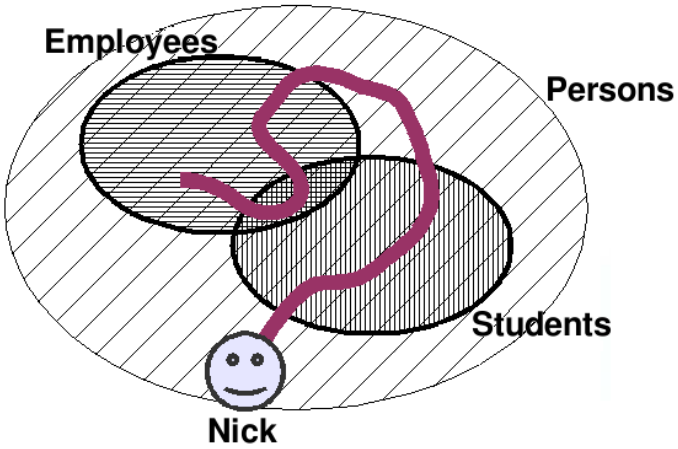

Figure 4.4: Dynamic classification [58]

denoted as *multiple classification*. In the previous example, in one moment, we were able to apply concepts *Person*, *Student* and *Employee* to object Nick. On the other hand, applicable concepts of some object change during their life-cycles. This change is denoted with *dynamic classification*. Whilst multiple classification is rather well supported by object-oriented programming languages (like Java and c++), dynamic classification is often not supported (python is an example that supports dynamic classification).

4.1.5 Relations and Mappings

Relations define the ways in which different objects can relate to each other. They also define certain communication channels for interaction between objects. By establishing such relations we are able to create conceptual networks.

Relations allow us to connect objects into families which we call tuples or links. Such tuples give us the opportunity to map objects of one type to objects of another type and vice versa. Tuples are immutable. For example if we have a tuple *(Sam,Microsoft)* which states that an object name Sam of type *Person* is employed by an object Microsoft of type *Organization* as shown on figure 4.5.

One cannot just remove the object Microsoft from the tuple and exchange it with another object of the same type (for example IBM), since the mapping has additional associated objects to it like a contract, a time frame or a salary. If changing to IBM these object wouldn't be the same. Thus the only thing one can do is to destroy the existing mapping and create a new one, but elements of tuples cannot be changed.

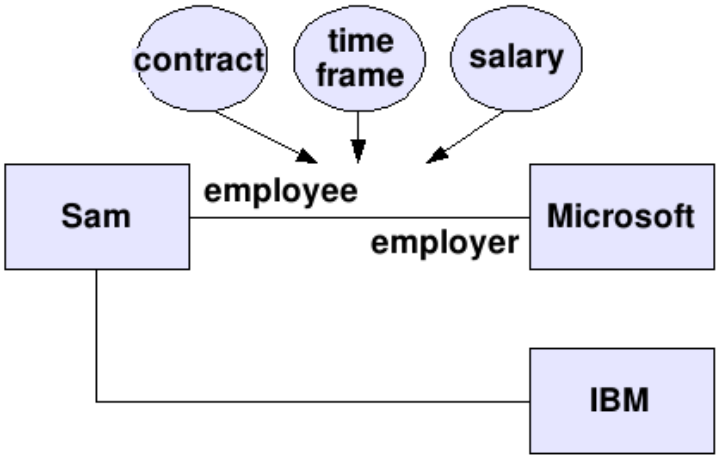

Figure 4.5: Relations and mappings [58]

4.1.6 Attributes and Attribute Values

It is sometimes pleasant to denote some relations to less complex objects as attributes. For example one would rather say that objects of type *Student* have attributes like name, surname, student-id, then say that students are related to objects of type string denoting their name, surname and student-id.

Thus a student would have attributes name, surname, student-id, and their particular instances John, Doe, and 32455-I would be the particular attribute values respectively.

4.2 The Semantic Web

The World Wide Web is made for humans. Humans can read, understand, as well as reason and draw conclusion from content encountered on the Web. But, as soon as one tries to automate reasoning about knowledge embodied in such content problems emerge.

For example, what if a user wants the answer to a simple question: "I need to see a doctor who is specialized in field X, who is relatively near to my house and has working hours in my lunch break from 12:00 - 14:00. Can you suggest one?" In order to answer such a question using the Web a regular user would probably sit in front of a computer, open up her favorite search engine and spend the next few hours searching and comparing data from different web sites.

The Semantic Web is an extension of the current (traditional) World Wide Web that shall allow one to find, share, and combine information more easily. It relies on machine-readable information and meta data that allows computer programs or agents to reason about distributed knowledge.

In a Semantic Web environment to return to our "doctor" example all doctors would have additional meta information attached to their web sites. The user would ask the question to an intelligent agent and go for a coffee break. The agent would do all the work for the end user providing him with the necessary information.

4.2.1 Ontologies

The machine-readable information or meta data is most often represented in special models called ontologies. One should make a distinction between ontology in philosophy (representing one of the fundamental branches in metaphysics concerned with existence) and ontology in information science (representing a formal domain model by introducing a set of concepts, instances, relations, axioms etc.).

Ontologies are an important part of the Semantic Web infrastructure as outlined in figure 4.6.

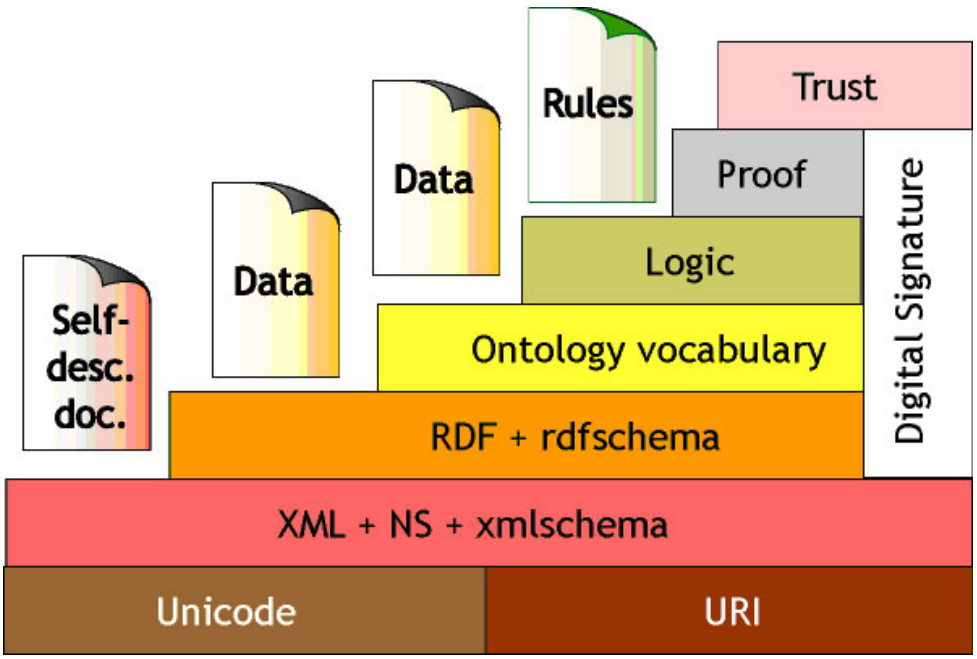

Figure 4.6: Semantic Web Stack [10]

Such ontologies are machine-readable structured data that allows some reasoning engine to draw conclusions from them without need human intervention.

4.2.2 Semantic Web Languages

In order to write semantic web ontologies and meta data one needs to provide a suitable language. Lots of such languages were proposed like XML, SHOE, XOL, OIL,

DAML+OIL, RDF, OWL, Flora-2, etc. In this thesis we will concentrate on OWL and especially Flora-2 since it is an implementation of frame logic.

Web Ontology Language

OWL or Web Ontology Language is a family of languages (OWL Lite, OWL DL and OWL Full) that is used for knowledge representation in a Semantic Web environment. OWL is considered to be one of the fundamental technologies for the Semantic Web and enjoys major attention in academia and industry. The World Wide Web Consortium (W3C) fully supports and facilitates OWL.

OWL has two different semantics: OWL DL and OWL Lite that are based on description logic, and OWL Full that is compatible with RDFS. From the beginning of the 1990s lots of research was conducted in order to find knowledge representation languages from the field of artificial intelligence that could be usable in the World Wide Web. Most were based on HTML (SHOE), XML (XOL and late OIL) as well as others. OWL was established through a revision of DAML+OIL which was used for Web ontologies.

As indicated above, the OWL specification supported by the W3C defines three versions of OWL: OWL Lite, OWL DL and OWL Full, whereby every version includes all conclusions of its predecessors.

OWL Lite is predominantly oriented towards users that need a class hierarchy and simple constraints. Nevertheless, the development of tools for OWL Lite proved to be as demanding as for the more complex OWL DL, which is why OWL Lite is rarely in use today.

OWL DL is designed to allow maximal expressiveness under the circumstances that (1) the derivation of all possible solutions is granted and (2) all derivations are done in finite time.

OWL Full uses a different semantics of the other two. For example OWL Full allows that a class is treated as a set of instances as well as an instance for it self, which OWL DL does not allow. OWL Full does not make any promises about derivation and finiteness [109].

FLORA-2

Frame logic or F-Logic is a full-fledged logic that has model-theoretic semantics and a sound and complete proof theory. In this sense, F-logic stands in the same relationship to the object-oriented paradigm as classical predicate calculus stands to relational program-

ming [41]. $\mathcal{F}$LORA-*2* is a rule-based object-oriented knowledge base system designed for a variety of automated tasks on the Semantic Web, ranging from meta-data management to information integration to intelligent agents. The $\mathcal{F}$LORA-*2* system integrates F-logic, HiLog, and Transaction Logic into a coherent knowledge representation and inference language which results in a flexible and natural framework that combines rule-based and object-oriented paradigms [116].

4.3 Web Services

Web services are modern networking technologies which enable remote procedures or services usage as if they were local. They enable the development of distributed networking applications without the need to contain all the parts of it on a single computer or server. Web services often use XML as a data description language (SOAP & WSDL) which are very simple and intuitive and it is often used in conjunction with web services for interchange of data between the local application and the service [8].

XML based web services can be used in any scenario that needs network-based exchange of information - from business transactions to simple exchange of news between web portals. To make this possible, web services are independent of operating system and programming language.

Web services are the logical next step in Internet development towards a loosely coupled structure that aims on exchanging the traditional heterogeneous one. This is why companies invest into web services development in order to prevent the incompatibility of existing data formats an application systems as well as to minimize costs for data exchange.

Business to business (B2B) communication is a modern communication concept between organizations, distinguishing it from Business to Customer (B2C) communication model used by the organizations to communicate with their customers. B2B often relies on concepts such as web services and XML for the interchange of data [8].

4.4 Semantic Web Services

The idea of semantic web services tries to combine web services with ideas from the Semantic web. The aim on a client-server system for machine-to-machine interaction through the Web. Semantic web services use meta data in form of markup which makes

them machine-readable in a detailed and sophisticated way.

Conventional XML based standards for the interoperability of web services only syntactically describe the exchange of messages, whilst the semantics isn't addressed at all. For instance, WSDL describes operations that are available on a given web service as well as the structure of messages. Still, it does not define the meaning of messages and operations nor the constraints above them. This makes automated interconnection of web services hard to implement. Semantic web services are build upon existing semantic web standards for semantic data interchange, which makes such an automated interconnection much easier [9].

4.5 Semantic Wiki Systems

Semantic wiki systems are an extension to wiki systems that use concepts borrowed from the semantic web. In this way semantics or meaning is added to knowledge created on the system through additional meta information which eases search, integration and reasoning [88]. Prominent semantic wiki systems include SweetWiki [98], IkeWiki [80] and Semantic Media Wiki [89] to name a few.

Semantic wiki systems have gained major attention of the academic community in the past few years. The idea of integrating the semantic web with fast and lightweight content management systems seems to be good direction towards web 3.0 or the social semantic web.

Chapter 5

Programming Languages for Semantic Wiki Systems

Most semantic wiki systems allow users to add semantics to content published on the system through different meta information. We decided to use a simple tagging system to allow users to approach content in an object-oriented manner. Whilst the idea of using semantic web concepts in wiki systems isn't new, the idea of using an object-oriented approach seems to be firstly described in [87]. Most other approaches used description logic's and particularly OWL as their background logic [80], [89], [98] as well as annotations and tags on particular content. The problem with such approaches lies mostly in ignoring the primary users of wiki as well as tagging systems.

Wiki systems are used by ordinary people who most certainly have no good understanding of semantic technologies. They use wikis to quickly create content in a collaborative environment. Tagging systems are also used by ordinary people to organize content they encounter for their selves in order to easier retrieve content. The semantics that emerge by combining tags of different users are more of a side effect than the main purpose. If users are free to tag any content in the way they want more meta information and semantics will emerge as when tagging is restricted to special types of tags that can be used. This meta information will self-organize due to the autopoiesis of the social system surrounding a dynamic web application like a tagging or a wiki system [8].

5.1 Frame Logic

We decided to use frame logic to define semantic wiki language. The alphabet $\Sigma_{\mathcal{F}}$ of an F-logic language $\mathcal{L}_{\mathcal{F}}$ consists of the following [41]:

- a set of object constructors, $\mathcal{F}$;
- an infinite set of variables, $\mathcal{V}$;
- auxiliary symbols, such as, (,), [,], $\rightarrow$, $\twoheadrightarrow$, $\bullet\!\!\rightarrow$, $\bullet\!\!\twoheadrightarrow$, $\Rightarrow$, $\Rightarrow\!\!\!\!\Rightarrow$, etc.; and
- usual logical connectives and quantifiers, $\vee$, $\wedge$, $\neg$, $\longleftarrow$, $\forall$, $\exists$.

Object constructors (the elements of $\mathcal{F}$) play the role of function symbols in F-logic whereby each function symbol has an arity. The arity is a non-negative integer that represents the number of arguments the symbol can take. A constant is a symbol with arity 0, and symbols with arity ≥ 1 are used to construct larger terms out of simpler ones. An id-term is a usual first-order term composed of function symbols and variables, as in predicate calculus. The set of all variable free or ground id-terms is denoted by $U(\mathcal{F})$ and is commonly known as Herbrand Universe. Id-terms play the role of logical object identities in F-logic which is a logical abstraction of physical object identities.

A language in F-logic consists of a set of formulae constructed out of alphabet symbols. The most simple formulae in F-logic are called F-molecules.

A molecule in F-logic is one of the following statements:

- An is-a assertion of the form $C :: D$ (C is a non-strict subclass of D) or of the form $O : C$ (O is a member of class C), where C, D and O are id-terms;
- An object molecule of the form O [a ';' separated list of method expressions] where O is a id-term that denotes and object. A method expression can be either a non-inheritable data expression, an inheritable data expression, or a signature expression:
  - Non-inheritable data expressions can be in either of the following two forms:
    - A non-inheritable scalar expression
      ScalMethod@$Q_1, ..., Q_k \rightarrow T$,$(k \geqslant 0)$.
    - A non-inheritable set-valued expression
      SetMethod@$R_1, ..., R_l \twoheadrightarrow \{S_1, ..., S_m\}$ $(l, m \geqslant 0)$.
  - Inheritable scalar and set-valued expression are equivalent to their non-inheritable counterparts except that $\rightarrow$ is replaced with $\bullet\!\!\rightarrow$, and $\twoheadrightarrow$ with $\bullet\!\!\twoheadrightarrow$.
  - Signature expression can also take two different forms:

* A scalar signature expression

  $\text{ScalMethod@}V_1, ..., V_n \Rightarrow (A_1, ..., A_r),\ (n, r \geqslant 0).$

* A set valued signature expression

  $\text{SetMethod@}W_1, ..., W_s \Rightarrow\!\!> (B_1, ..., B_t)\ (s, t \geqslant 0).$

All methods' left hand sides (e. g. Q_i, R_i, V_i and W_i) denote arguments, whilst the right hand sides (e. g. T, S_i, A_i and B_i) denote method outputs. Single-headed arrows ($\rightarrow$, $\bullet\!\!\rightarrow$ and $\Rightarrow$) denote scalar methods and double-headed arrows ($\rightarrow\!\!>$, $\bullet\!\!\rightarrow\!\!>$ and $\Rightarrow\!\!>$) denote set-valued methods.

As in a lot of other logic, F-formulae are built out of simpler ones by using the usual logical connectives and quantifiers mentioned above.

- F-molecules are F-formulae;
- $\varphi \vee \psi$, $\varphi \wedge \psi$, and $\neg\varphi$, are F-formulae if so are φ and ψ;
- $\forall X \varphi$ and $\exists Y \psi$ are F-formulae, so are φ and ψ, and X and Y are variables.

For our purpose these definitions of F-logic are sufficient but the interested reader is advised to consult [41] for profound logical foundations of object-oriented and frame based languages.

5.2 Semantic Wiki Language

A semantic wiki language $\mathcal{L}_{SW}$ is an addition to a wiki language $\mathcal{L}_W$ that allows the definition, manipulation and querying of meta data in form of a knowledge base. Thus a semantic wiki language consists of two parts: (1) a wiki component (already defined in chapter 3) and (2) a semantic component. In the following we shall use frame logic or F-logic to formalize this semantic component of semantic wiki languages.[1] The basic idea is to map concepts from wiki systems to concepts from frame logic.

In order to consider a domain of interest in an object-oriented fashion one needs to be able to model specific concepts like objects, classes (types, concepts), relations, attributes, methods, states etc. Thus we provide the following conceptualization of semantic wiki systems.

Let the whole content stored on the semantic wiki system be a *domain* of interest D. Objects inside this domain are specific wiki pages having their classes, relations,

[1] Off course there are other possibilities like the mentioned description logic approach.

attributes, methods etc. Any wiki page on creation is a generic object that users can specialize in order to reflect the domain of interest. Thus the domain is an extensible set of objects as shown in tho following equation.

$$D = \{o_1, o_2, ..., o_n\}$$

To allow concretization of generic objects we introduce attribute-value tags that reflect specific characteristics of objects inside a domain. Any object can be thought of as a relation that consists of a finite number of attribute-value tuples, as shown in the following equation.

$$\text{att}(o_i) = \{(a_1, v_1), (a_2, v_2), ..., (a_m, v_m)\}$$

This set also includes standard attributes like author(s), title, body (written in $\mathcal{L}_{\mathcal{W}}$) etc.

We also introduce object's relations to be defined as labeled outgoing links on any wiki page whether to other wiki pages or to pages outside the semantic wiki system. These relations are reflected as additional attribute-value pairs whereby the label represents the attribute and the value the object (page or URL)[2] the relation links to. Thus the set of an objects outgoing relations is shown in the following equation.

$$\text{rel}(o_i) = \{(r_1, o_{r1}), (r_2, o_{r2}), ..., (r_l, o_{rl})\}$$

In the end we introduce a set of methods represented through web services or script extensions of the form $(\text{m}_i(p_{i1}, p_{i2}, ..., p_{ia_i}), res_i)$ where m is the methods name, $p_{i1}, ..., p_{ia_i}$ are the methods parameters, a_i is the methods arity, and res_i is the methods return value. These methods are represented through the set:

$$\begin{aligned}\text{met}(o_i) = \{ \quad & (\text{m}_1(p_{11}, p_{12}, ..., p_{1a_1}), res_i), \\ & (\text{m}_2(p_{21}, p_{22}, ..., p_{2a_2}), res_2), \\ & \vdots \\ & (\text{m}_l(p_{k2}, p_{k2}, ..., p_{ka_k}), res_k) \quad \}\end{aligned}$$

Definition Let att(o) be the set of attribute-value pairs of object o, rel(o) the set of

[2]Unified Resource Locator

relation-object's identifier pairs of object o, and met(o) the set of methods-return value pairs. An *object* with id-term o in a semantic wiki language $\mathcal{L}$ then is represented with the F-molecule:

$$
\begin{aligned}
o[& \\
& a_1 \rightarrow v_1; \\
& a_2 \rightarrow v_2; \\
& \vdots \\
& a_m \rightarrow v_m; \\
& r_1 \rightarrow o_{r1}; \\
& r_2 \rightarrow o_{r2}; \\
& \vdots \\
& r_l \rightarrow o_{rl}; \\
& \mathrm{m}_1(p_{11}, p_{12}, ..., p_{1a_1}) \Rightarrow res_1; \\
& \mathrm{m}_2(p_{21}, p_{22}, ..., p_{2a_2}) \Rightarrow res_2; \\
& \vdots \\
& \mathrm{m}_k(p_{k1}, p_{k2}, ..., p_{ka_k}) \Rightarrow res_k \\
&].
\end{aligned}
$$

Such a definition implies that attribute-value tags (a_i, v_i) associated with a given wiki page (or object) with id-term o, as well as relation-object's identifier pairs are considered to be *non-inheritable scalar methods* whereby the attribute (a_i) is the methods name that has no arguments $(k = 0)$ and values (v_i) to be outputs. If there are more than one equivalent attributes or relation names for a given object with distinct values than the method is considered to be a *non-inheritable set-valued method*. In the end signature expressions are considered to be web services and script extensions that act as methods of a specific object.

Now we are able to introduce special attributes labeled with common object-oriented programming constructs like `class`, `subclass`, `rule` etc. Such attributes are used to provide additional semantics to the domain ontology. Special attribute labels like `class` and `subclass` are used to create is-a assertions. For example a wiki page tagged with the

attribute `class` and value `student` is considered to be an object that is a member of class `student`. If the same object is additionally tagged with the attribute `subclass` and value `person` than the class `student` is considered to be a non-strict subclass of class `person`. All other tags provided on a wiki page are the special attributes of this object, thus the object mentioned previously if additionally tagged with tags like `name:Foo`, `surname:Bar`, `address:Linus Lane 27` would yield the following sentence in a F-logic knowledge base:

$$\begin{aligned} student &:: person \wedge \\ o_x &: student\ [\\ name &\rightarrow 'Foo'; \\ surname &\rightarrow 'Bar'; \\ address &\rightarrow 'Linus\ Lane\ 27'\ ] \end{aligned}$$

Where o_x denotes the logical object-id of the wiki page under consideration. Thus, classes and class hierarchy are created dynamically by tagging specific objects.

Definition An object with id-term o is considered to be a member of class c if its corresponding F-molecule contains a non-inheritable scalar method class$\rightarrow c$.

Definition A class c_1 is considered to be a non-strict subclass of class c_2 if there is at least one object that is a member of class c_1 which corresponding F-molecule contains a non-inheritable scalar method subclass$\rightarrow c_2$.

Definition A given object with id-term c, which corresponding F-molecule contains the attribute `class` which value is also `class`, is considered to be a class descriptor. All its attribute-value pairs are converted to inheritable scalar expressions (scalar or set-valued depending on the number of equivalent attributes with distinct values) except for the `class:class` pair.

In this way we allow for meta modeling, by stating that instances of class `class` are classes of their own. For instance if a wiki page entitled `car` is tagged with the tags `class:class`, `model:string`, `color:string`, and `year:integer` it would correspond to the following sentence in frame logic:

$$
\begin{aligned}
car[& \\
model & \bullet\!\!\rightarrow string; \\
color & \bullet\!\!\rightarrow string; \\
year & \bullet\!\!\rightarrow integer]
\end{aligned}
$$

Such interpretation allows us then to create instances of such a defined class as well as to define the schema of a domain of a given wiki system.[3]

Another important concept is the definition of rules. Rules are defined in terms of objects tagged with special attribute `rule`).

Definition If some object is tagged with special attribute `rule`, and value of the form `Head :- Body` then this attribute-value pair is removed from the object descriptor and the following rule is added to the knowledge-base:

$$Head \leftarrow Body$$

For instance if some wiki page was tagged with `rule : ?x:boy :- ?x:person[ sex->male, age->?a ], ?a<18` the following rule would be added to the knowledge-base:

$$?x : boy \leftarrow ?x : person[\ sex \rightarrow male\ ;\ age \rightarrow ?a\ ] \vee ?a < 18$$

Since we were able to map concepts from semantic wikis to concepts from F-logic we can now state that the syntax of the semantic component of semantic wiki languages is equivalent to the syntax of F-logic defined above.

5.2.1 Semantic Wiki Syntax

The following grammar defines the actual syntax of a semantic wiki language.[4] A semantic wiki page consists of a set of statements and (eventually) of additional meta information.

[3] The schema (defined or inferred) is used in ┬AOPīs for input suggestion mechanisms. Such mechanisms try to minimize syntactic errors due to different user input. For a better understanding of such input mechanisms please refer to [86]

[4] For a complete XSB Prolog implementation of a niKłąs semantic wiki syntax parser please refer to appendix B.

```
<semantic_wiki_page> ::= <statement>* <metainfo>? ;
<statement> ::= <STRING>
              | <formatting_expression>
              | <display_object>
              | <comment>
              | <hyperlink>
              | <table>
              | <variable_template>
              | <reference_entry>
              | <reference_citation>
              | <query>
```

The possible statements were already defined in chapter 2 except for the `<query>` statement which we shall define in the following few sections. Prior to that we need to define a crucial concept that is an important part of any query - namely semantic templates.

5.2.2 Semantic Templates

A semantic template is defined with the following simple production rule in EBNF.

```
<semantic_template> ::= ( <statement> | <frame_logic_variable> )*
```

By using regular expressions this statement can be defined as.

Definition Let r_{variable} be a regular expression that matches all possible variables in a F-logic language defined over Σ, and r_W a regular expression that defines a wiki language $\mathcal{L}_{\mathcal{W}}$. A semantic template with the regular expression:

$$r_{\text{semantic template}} = r_W | r_{\text{variable}}$$

This definition needs further explanation. We consider a wiki page to be a collection of letters (wiki text) which is interpreted using concepts from a wiki language $\mathcal{L}_W$. We now introduced a new concept (through a new regular expression) that will act as a variable. The actual value to which this value will be bound depends on the context in which the template is used. The context will probably be associated with a given query,

defined later on. Thus a semantic template is a placeholder, that will yield wiki text when all its variables are exchanged with actual values from the knowledge base.

For example the niKlas syntax for for variables is equivalent to the $\mathcal{F}$LORA-*2* syntax for logic variables, e.g.:

$$r_{\text{variable}} \quad = \quad \backslash?[\text{a-zA-Z0-9\_\$ }]^*$$

Thus the following is a valid semantic template in niKlas :

```
[link=?url>?link_name]
```

5.2.3 Queries

Queries are closely bound to frame logic syntax which makes its set of production rules more complicated. The following listing shows the grammar dealing with queries:[5]

```
<query> ::=
  <query_start>
  <frame_logic_query>
  <semantic_template>
  <query_end>

<frame_logic_query> ::=
  <frame_logic_rule_body>

<frame_logic_rule_body> ::=
  <frame_logic_list_of_literals>

<frame_logic_rule_head> ::=
  <frame_logic_list_of_molecules>

<frame_logic_list_of_molecules> ::=
  <frame_logic_molecule>
```

[5]The production rules for frame logic syntax were taken from [61]. The lexical structure is given in appendix B. In the actual implementation, due to the fact that TaOPIs uses $\mathcal{F}$LORA-*2* syntax, the rules were adjusted to allow for HiLog extensions and other syntactic differences.

```
  ( ',' <frame_logic_list_of_molecules> )?

<frame_logic_list_of_literals> ::=
  <frame_logic_literal>
  ( ',' <frame_logic_list_of_literals> )?

<frame_logic_literal> ::= 'not'? <frame_logic_molecule>

<frame_logic_molecule> ::=
    <frame_logic_fmolecule>
  | <frame_logic_pmolecule>

<frame_logic_pmolecule> ::=
  <frame_logic_predicate>
  ( '(' <frame_logic_list_of_expressions> ')' )?
 | <frame_logic_built_in_predicate>
  ( '(' <frame_logic_list_of_expressions> ')' )?
 | <frame_logic_arithmetic_expression>
   <frame_logic_infix_built_in_predicate>
   <frame_logic_arithmetic_expression>

<frame_logic_list_of_expressions> ::=
  <frame_logic_expression>
  ( ',' <frame_logic_list_of_expressions> )?

<frame_logic_expression> ::=
    <frame_logic_path_expression>
  | <frame_logic_fmolecule>
  | <frame_logic_aggregrate>

<frame_logic_arithmetic_expression> ::=
    <frame_logic_expression>
  | <frame_logic_arithmetic_expression>
    <frame_logic_built_in_operator>
```

```
    <frame_logic_arithmetic_expression>
  | '(' <frame_logic_arithmetic_expression> ')'

<frame_logic_aggregrate> ::=
  <frame_logic_id_term>
  '{'
    <frame_logic_variable>
    ( '[' frame_logic_list_of_variables ']' )?
    ';'
    frame_logic_list_of_literals
  '}'

<frame_logic_fmolecule> ::=
  <frame_logic_path_expression>
  <frame_logic_specification>

<frame_logic_path_expression> ::=
    <frame_logic_id_term>
  | '(' <frame_logic_expression> ')'
  | <frame_logic_path_expression>
    <frame_logic_dot>
    <frame_logic_method_application>
  | <frame_logic_fmolecule>
    <frame_logic_dot>
    <frame_logic_method_application>

<frame_logic_specification> ::=
    <frame_logic_is_a_specification>
    '[' <frame_logic_list_of_methods>? ']'
  | <frame_logic_is_a_specification>
  | '[' <frame_logic_list_of_methods>? ']'

<frame_logic_is_a_specification> ::=
    <frame_logic_is_a_symbol>
```

```
    <frame_logic_id_term>
  | <frame_logic_is_a_symbol>
    '(' <frame_logic_expression> ')'

<frame_logic_method_application> ::=
    <frame_logic_id_term>
    ( '@(' <frame_logic_list_of_expressions> ')' )?
  | '(' <frame_logic_expression> ')'
    ( '@(' <frame_logic_list_of_expressions> ')' )?

<frame_logic_list_of_methods> ::=
  <frame_logic_method_application>
  <frame_logic_method_result>
  ( ';' <frame_logic_list_of_methods> )?

<frame_logic_method_result> ::=
    <frame_logic_method_arrow1>
    <frame_logic_expression>
  | <frame_logic_method_arrow2>
    <frame_logic_expression>
  | <frame_logic_method_arrow2>
    '{' <frame_logic_list_of_expressions> '}'
  | <frame_logic_method_arrow3>
    <frame_logic_expression>
  | <frame_logic_method_arrow3>
    '{' <frame_logic_list_of_expressions> '}'

<frame_logic_id_term> ::=
    <frame_logic_basic_id_term>
  | <frame_logic_functor>
    '(' <frame_logic_list_of_expressions> ')'

<frame_logic_basic_id_term> ::=
    <frame_logic_functor>
```

```
  | <frame_logic_variable>
  | <frame_logic_string>
  | <frame_logic_integer>

<frame_logic_list_of_variables> ::=
  <frame_logic_variable>
  ( ',' <frame_logic_list_of_variables> )?
```

Definition Let the alphabet of wiki language $\mathcal{L}_W$ be a superset of F-logic alphabet $\Sigma_{\mathcal{F}}$. Let further q_{begin} be a regular expression that matches all query words beginnings, q_{formula} be a regular expression that matches possible F-logic formulas,[6] $q_{\text{delimiter}}$ a regular expression that matches delimiter words, $r_{\text{semantic template}}$ a regular expression that matches semantic templates, and q_{end} be a regular expression that matches all query word's endings. Let the following set of relations hold:

$$q_{\text{formula}} \notin q_{\text{delimiter}}$$
$$r_{\text{semantic template}} \notin q_{\text{end}}$$

Then a *query* is defined with the following regular expression:

$$r_{query} = q_{\text{begin}} q_{\text{formula}} q_{\text{delimiter}} r_{\text{semantic template}} q_{\text{end}}$$

The semantics of a query read as follows: for each result result_i obtained by issuing the query defined by q_{formula} against the knowledge base of the wiki system (the domain D) interpret the semantic template defined by $r_{\text{semantic template}}$ by exchanging any occurrence of a variable with the corresponding value from result_i. In a semantic wiki context this means that if on a wiki page a query occurs, than the formula defined by the query will be issued as a query against the knowledge base defined by the meta data of the wiki system. Each result will force the wiki language interpreter to write the wiki text of the semantic wiki template by exchanging all variables in it with corresponding values obtained from the result.

[6] Herein we leave the possibility open if this regular expression will possibly match words that aren't F-logic formulas since frame logic is more expressive than regular expressions.

The **niKlas** syntax for semantic queries is defined as follows:

$$
\begin{aligned}
q_{\text{begin}} &= [\text{query}= \\
q_{\text{formula}} &= .* \\
q_{\text{delimiter}} &=]\backslash n \\
r_{\text{semantic template}} &= ([/\text{query}])!* \\
q_{\text{end}} &= \backslash n[/\text{query}]
\end{aligned}
$$

For example a query that is used on a wiki dealing with our courses dealing with databases there are a lot of different examples. To create a dynamic list of existing examples we use a query similar to the following:[7]

```
[h2]Examples[/h2]
[query=
  ?_:wiki_page[
    example->?title ,
    url->?url ],
  sort(?title ,asc). ]
* [link=?url>?title]
[/query]
```

The query generates a list of links to examples as shown on figure 5.1

If we would have used the semantic wiki parser from appendix B the following parse tree would be obtained:

```
semantic_wiki_page(
 statements(
  [
   statement(
    formatting_expression(
     formatting_expression_start(
      [h2]
     ),
```

[7]The query is translated to English for the purposes of this thesis.

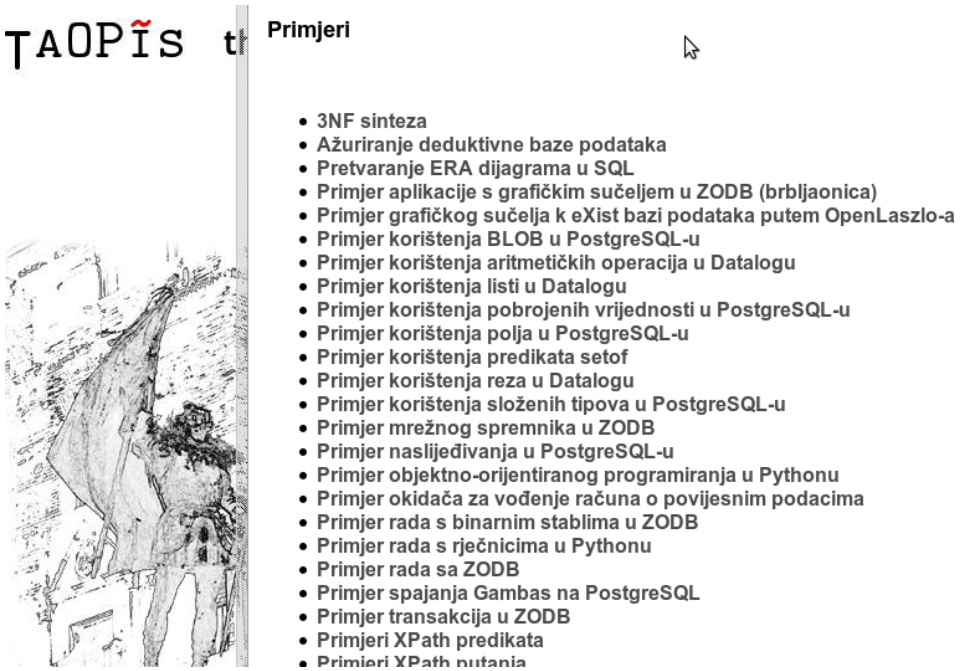

Figure 5.1: List of examples generated by a query

```
    statements(
     [
      text(
       Examples
      )
     ]
    ),
    formatting_expression_end(
     [/h2]
    )
   )
  )
 ),
 statement(
  query(
   query_start(
    [query=,
    flogic_query(
     flogic_rule_body(
      flogic_list_of_literals(
       [
```

```
flogic_literal(
 flogic_molecule(
  flogic_fmolecule(
   flogic_path_expression(
    flogic_id_term(
     flogic_basic_id_term(
      flogic_variable(
       ?_
      )
     )
    )
   ),
   flogic_specification(
    flogic_isa_specification(
     flogic_isa_symbol(
      :
     ),
     flogic_id_term(
      flogic_basic_id_term(
       flogic_functor(
        wiki_page
       )
      )
     )
    ),
    flogic_list_of_methods(
     [
      flogic_method_specification(
       flogic_method_application(
        flogic_id_term(
         flogic_basic_id_term(
          flogic_functor(
           example
          )
```

```
    )
   )
  ),
  flogic_method_result(
   flogic_method_arrow1(
    ->
   ),
   flogic_expression(
    flogic_path_expression(
     flogic_id_term(
      flogic_basic_id_term(
       flogic_functor(
        flogic_variable(
         ?title
        )
       )
      )
     )
    )
   )
  )
 ),
 flogic_method_specification(
  flogic_method_application(
   flogic_id_term(
    flogic_basic_id_term(
     flogic_string(
      url
     )
    )
   )
  ),
  flogic_method_result(
   flogic_method_arrow1(
```

```
          ->
         ),
         flogic_expression(
          flogic_path_expression(
           flogic_id_term(
            flogic_basic_id_term(
             flogic_variable(
              ?url
             )
            )
           )
          )
         )
        )
       )
      ]
     )
    )
   )
  )
 ),
 flogic_literal(
  flogic_molecule(
   flogic_pmolecule(
    flogic_predicate(
     sort
    ),
    flogic_list_of_expressions(
     [
      flogic_expression(
       flogic_path_expression(
        flogic_id_term(
         flogic_basic_id_term(
          flogic_variable(
```

```
                ?title
               )
              )
             )
            )
           ),
           flogic_expression(
            flogic_path_expression(
             flogic_id_term(
              flogic_basic_id_term(
               flogic_functor(
                asc
               )
              )
             )
            )
           )
          ]
         )
        )
       )
      )
     ]
    )
   )
  )
 ),
 semantic_template(
  statements(
   [
    text(
     *
    ),
    statement(
```

```
          hyperlink(
           hyperlink_start(
            [link=
           ),
           internal_url(
            flogic_variable(
             ?url
            )
           ),
           link_name(
            flogic_variable(
             ?title
            )
           ),
           hyperlink_end(
            ]
           )
          )
         )
        ]
       )
      ),
      query_end(
        [/query]
      )
     )
    )
   ]
 )
)
```

Another interesting query for this particular wiki is the query that generates a list of programming languages used on different wiki pages.

```
[h2]Programming languages[/h2]
```

```
[table]
[query=
  ?_:wiki_page[
    'programming language'->?name ],
  sort(?name,asc). ]
[link=http://autopoiesis.foi.hr/tag.php??search=yes&attribute=
→programming language&value=?name>?name] | [/query]
[/table]
```

This query yields a tag cloud of programming languages (figure 5.2), whereby each link points to a list of pages using the particular programming language.

Figure 5.2: Tag cloud of programming languages generated by a query

5.2.4 Meta Information

Meta information is defined with the following set of production rules:

```
<metainfo> ::= ( <attribute_value_tag> | <hyperlink> )*
<attribute_value_tag> ::= <attribute> <value>
                        | 'class' <frame_logic_id_term>
                        | 'subclass' (
                              <frame_logic_id_term>
                            | '(' <frame_logic_expression ')' )
                        | 'rule' <frame_logic_rule_head> :-
                                 <frame_logic_rule_body>
<attribute> ::= <frame_logic_id_term>
<value> ::= <frame_logic_id_term>
<internal_url> ::= <frame_logic_id_term>
<external_url> ::= <frame_logic_id_term>
```

Definition Let $r_{\text{attribute-value tag}}$ be a regular expression that matches attribute-value tags, and $r_{\text{hyperlink}}$ be a regular expression that matches hyperlinks, then the following regular expression matches meta information.

$$r_{\text{meta information}} = (r_{\text{attribute-value tag}} | r_{\text{hyperlink}})*$$

We are now able to define the semantic wiki language $\mathcal{L}_{SW}$ as follows:

Definition Let $\mathcal{L}_W$ be a wiki language, $\mathcal{L}_\mathcal{F}$ a F-logic language as defined above, let r_{query} be a regular expression that defines queries, and let $r_{\text{meta information}}$ be a regular expression that matches meta information. A *semantic wiki language* $\mathcal{L}_{SW}$ is the pair $(\mathcal{L}_W, \mathcal{L}_\mathcal{F})$ bridged through r_{query} and $r_{\text{meta information}}$. $\mathcal{L}_W$ is called the wiki component of language $\mathcal{L}_{SW}$, and $\mathcal{L}_\mathcal{F}$ is called the semantic component. r_{query} and $r_{\text{meta information}}$ are the interface between $\mathcal{L}_W$ and $\mathcal{L}_\mathcal{F}$.

5.3 Inconsistencies in Semantic Wiki Systems

The previous definitions leave some questions open as outlined in [86]. The ŤAOPĪS system has been used for almost two years for various projects including open-source project management, university course documentation, political activism, alumni, job search etc. After analyzing the meta data provided by its users, we were able to observe basically two types of inconsistencies:

- **Syntactical inconsistencies** - arisen mostly due to different or inadequate spelling in attribute-value tags;
- **Semantic inconsistencies** - arisen mostly due different views of project members.

While syntactical inconsistencies can be easy detected, solved and prevented, semantic inconsistencies can yield problems in various situations. For example in a case there was a cyclic definition of subclass relations depicted on figure 5.3. Such a subclass definition can pose problems for intelligent agents reasoning about this particular domain.

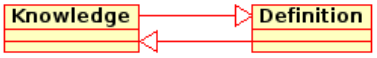

Figure 5.3: Inconsistent definition of subclasses

Other examples include multiple class definitions for a given object (page), as well as set relations to one and the same object due to multiple hyperlinks to the same page.

To minimize and prevent syntactic errors a suggestion mechanism for attribute-value tags was implemented. The suggestion mechanism allows the user to see the possible classes in a specific domain, attributes for a given class as well as attribute values for a given attribute name. Such a suggestion mechanism allows for less syntactical inconsistencies. The new entry form is shown on figure 5.4.

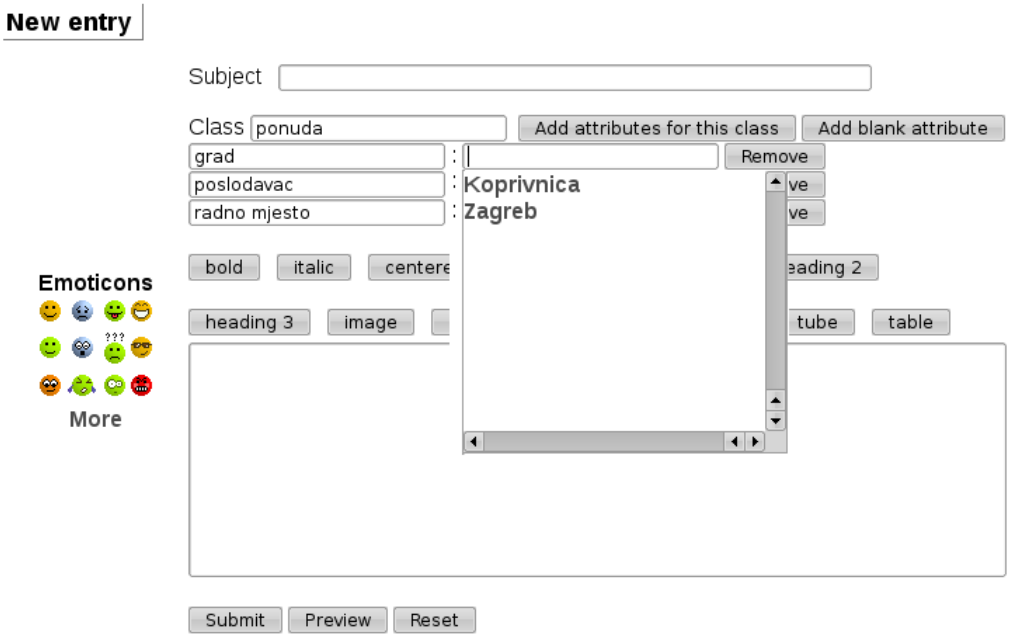

Figure 5.4: Suggestion mechanism entry form

As shown on the figure when adding a specific attribute or class the user gets a suggestion of the system for similar terms. On the image the user already entered the attribute city (cro. *grad*) and the system automatically shows possible values for this attribute (e.g. Koprivnica and Zagreb).

Semantic inconsistencies are much harder to prevent, which is why it is suggested to couple to the social system. Social network analysis allows to detect the most trustworthy members of a social network. In ŦAOPĪS a special case of eigenvector centrality [11] used in the PageRank algorithm [72]. Users can vote for each other on a given project to establish the trust network. In this way numerical values of trust can be obtained and used to annotate meta information. Before dealing with the details of this annotation scheme, an introduction to autopoiesis and autopoietic systems shall be given.

Chapter 6

Autopoiesis and Autopoietic Systems

6.1 Introducing Autopoietic Systems

This chapter summarizes the notion of autopoietic information systems as subsystems of organizations and social systems in a broader perspective. Different notions of autopoiesis in biology, sociology and organization theory are analyzed in order to yield a definition of autopoietic information systems. Modern organizational approaches are described using autopoietic theory in order to be supported by information systems. In the end the ᴛAOPĪS system is described as well as possible application of autopoietic information systems.

We shall try to engage autopoietic information systems starting at their very beginning, their theoretical foundations and ending with practical issues we encountered during our research. The main idea is to try to answer the fundamental question what an autopoietic information system is? Is it possible to apply autopoiesis to systems traditionally considered to be alopoietic? How to support autopoietic information systems through information and communication technology? Where lies the main usage of such systems?

We try to give theoretical foundations that are necessary and crucial to any discussion about autopoietic information systems. Starting at the definitions of autopoiesis given in different scientific fields like biology, sociology and organization theory we try to develop a full definition of autopoietic information systems taking the requirements of modern organizations as well as modern information and communication technologies into consideration.

Modern organizational approaches like network organizations or heterarchies, virtual organizations, the hypertext organization, organizational architecture and other holistic views of an organization like the fractal company as well as process oriented

approaches that emerged in the last two decades let us sense a new paradigm in organizational theory. Is it possible that the common denominator of these approaches is autopoiesis?

If we presume that the answer to this question is at least positive, is there a way to support this paradigm through modern information and communication technologies? During the evolution of the World Wide Web a lot of new technologies emerged that showed amazing success in employing individuals creative powers, communication skills as well as collaborative techniques for the achievement of common goals and good. Technologies like forums, wiki systems, the semantic web, pod-casting, social networking, content feeds, tremendous search engines, the open-source paradigm, peer to peer networks, and others often commonly denoted with the term Web 2.0 or Web 3.0 seem to be a good platform for attaining this goal. Is it possible to employ these technologies in modern organizations, and if yes how?

Having such a reasoning in mind we tried to implement a system that could answer to the questions given above. In this chapter we are documenting our experiences gathered during the development of the ŦAOPĪS project. In the end we give guidelines and forecasts for the use of such systems in the practice of modern organizations as well as examples of applications that are possible.

6.2 Various Aspects of Autopoiesis

Autopoiesis, a pseudo Greek word coined from $\alpha\upsilon\tau\acute{o}$ (auto) for self and $\pi o\acute{\iota}\eta\sigma\iota\varsigma$ (poiesis) for creation, production or forming was first coined by the Chilean biologists Humberto Maturana and Francisco Varela in 1973 [60] to label the type of phenomenon which they had identified as the definitive characteristic of living systems [114].

Using the metaphor of autopoiesis a whole theory of social systems based on communication was developed later by Niklas Luhmann [53]. He introduced the concept of autopoiesis to formal organization theory basing his reasoning on a special subset of communication: decisions that, following Luhmann, are the essence of organization [56].

This three distinct conceptualizations of autopoiesis are different and in some cases incompatible as we shall show later in this chapter. In the following we shall give a brief overview of the different views on autopoiesis. Prior to that we need to make a clear distinction of two basic concepts, since they are used in the same context.

First there is the concept of organization that is used three-ways: (1) orga-

nization in a institutional sense – denoting a system of consciously coordinated peoples activities with a common goal [108, p. 5], (2) organization in Maturana's and Varela's sense – denoting the instrumental participation of components in the constitution of a unity [59, p. 315] or basically a system of relations that build up a unity and (3) organization in Luhmann's sense – denoting a system of decisions [55, p. 106].

As second there is the concept of structure that is used two-ways: (1) structure in the sense of (traditional) organizational structure – denoting a system of relations between organizational units as well as (2) structure in the sense of Maturana and Varela – denoting the medium upon which the organization (in Maturana's and Varela's sense) of a unity functions. To prevent possible confusion we shall use the terms organization and structure in their traditional senses if not stated otherwise.

As mentioned before the concept of autopoiesis was first introduced by Maturana and Varela to characterize living systems, as opposed to any other system. The original idea was to develop a new perspective of perception and cognition stating that cognition is a phenomenon of the living. Thus it was necessary to find out what characterizes living systems which lead to the notion of autopoiesis that became the core of the new perspective [114].

Varela gave the following definition of autopoietic systems:

"An autopoietic system is organized (defined as a unity) as a network of processes of production (transformation and destruction) of components that produces the components that:

1. *through their interactions and transformations continuously regenerate and realize the network of processes (relations) that produced them; and*

2. *constitute it (the machine) as a concrete unity in the space in which they [the components] exist by specifying the topological domain of its realization as such a network."* [102, p. 13] adapted from [113]

Maturana stated that "*... autopoietic systems operate as homeostatic systems that have their own organization as the critical fundamental variable that they actively maintain constant.*" [59, p. 318]. Thus the concept of autopoiesis involves organizational preservation and componential (re-)production [114].

According to Luhmann social systems are meaning processing systems and this is what distinguishes them form other types of systems such as biological ones [65, p. 104]. "*A social system comes into being whenever an autopoietic connection of communications*

occurs and distinguishes itself against an environment by restricting the appropriate communications. Accordingly, social systems are not comprised of persons and actions but of communications." [54, p. 145]. Social systems are networks of communication that produce further communication and only communication and are thus autopoietic systems [65, pp. 104–105].

Luhmann argues that there are three types of social systems: interactional, organizational and societal which differ mostly in terms of the ways they constitute themselves as well as they select and form their boundaries. Interactional systems are comprised of communication between a set of people by making a distinction between people one talks with and people one talks about. Societal systems do not rely only on communication taking place, but also on previous (stored) communication. Organizational systems are special since they are formed of a special type of communication – decisions that shape the possible future states of the system.

As one can see from these various aspects there are a few crucial concepts one should have in mind before any discussion about autopoiesis. First there is a distinction between structure and organization (in Maturana's and Varela's sense). While structure is something that is visible (observable) from the outside, organization is unobservable and inside of the system. Structure comprises of a set of components or elements that are exchangeable (which means that components change during time) and the mutual interactions between these components. Organization comprises of the relations between these components and is stable over time. That means that structure does change but organization remains stable even if the components that make up the structure change over time due to interaction of the system with its environment.

These connection between an autopoietic system and its environment is denoted as structural coupling (shown on figure 6.1.).[1] "*The result of structural coupling is an autonomous and strictly bounded system, that has nevertheless been shaped extensively by its interactions with its environment over time, just as the environment has been shaped by its interactions with the system.*" [75]

The mechanics of the process of autopoiesis as described by Maturana and Varela are kept strictly within the bounds of an autopoietic system. Thus autopoietic systems are closed in terms of operational and organizational closure [75].

While in living systems structure is comprised of biological processes in social systems structure is according to Luhmann comprised of communication. Organization

[1] With friendly reprint permission of Tom Quick

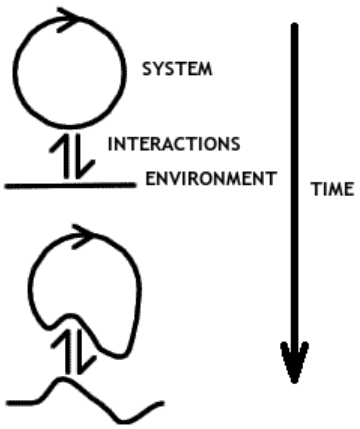

Figure 6.1: Structural Coupling [75]

(in Maturana's and Varela's sense) is then comprised of the particular relations between certain communicative events.

Another important concept is the reproduction of components. While one can easily depict this process in living systems (e.g. living beings feed themselves with food from their environment that eventually after certain processes becomes an integral part of the living being facilitating the regeneration of the process) in social systems this reproduction is less obvious. If we follow Luhmann then communicative events are reproduced by previous communicative events, or in the case of organizations (in Luhmann's sense) decisions reproduce new decisions.

To picture autopoiesis at a most basic level we could introduce an imaginary autopoietic system consisting of only one process and only one component. The process uses the resources from the component to produce new resources which in turn enable the recreation of the process. Thus the process' recursive relation with itself represents the organization and the component the structure of the system. This most basic autopoietic system is depicted on figure 6.2. whereby P1 represents the process and R1 the component. The resources in the component can but do not have to be from the environment.

6.3 Invitations to an New Paradigm

In the last 20 years one was able to observe a lot of new concepts in organizational theory. Terms like heterarchies, fishnet organizations, hypertext organizations, virtual organizations or fractal companies are often hard to classify using traditional organization theory concepts. In the following we give a brief description of important ideas.[2]

[2]You can find a more in depth discussion on modern approaches to organizations in [105].

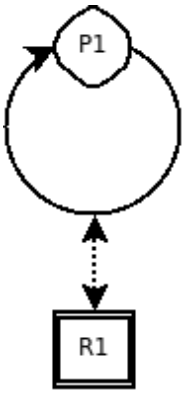

Figure 6.2: The basic autopoietic system

6.3.1 Heterarchies and the Fishnet Organization

The concept of a heterarchic organization (or network organization) is based on the following principles: an organization consisting of organizational units[3] that are mutually connected through information links (often based on modern information technology), are mutually independent, heterarchically organized (as opposed to hierarchy), and they operate internally and externally (with their environment) in most cases sharing some common goal [105, p. 106].

The idea of a heterarchical organization comes from the neuropsychological research of the human brain conducted by Warren McCulloch in 1945. He concluded that the human brain must have a heterarchical organization as opposed to previously defined hierarchical models, and described this organization as a neural network which is specifically designed for parallel information processing [78, p. 3].

If we apply such a concept to an organization, we get a structure which interrelationships are not strictly defined, but rather activated, or self regulated depending on the particular situation [105, p. 106].

An interesting metaphor for this kind of organization is the fishnet organization, depicted on figure 6.3. If we observe a fisher's net on the coast, it seems completely non-hierarchical. But if we take one node and lift it up, we get a hierarchical structure. By lifting further nodes and putting down the old ones, we can see the dynamical creation of new and the destruction of old hierarchical structures. Thus the fishnet organization tries to combine the modern concept of heterarchy and the usual human habit of tendency to hierarchy and order [88].

[3]Organizational units can in this context be either individuals, teams, departments, divisions and even entire organizations or groups of organizations by the fractal organization principle [105, pp. 149–151] as argued further in this chapter

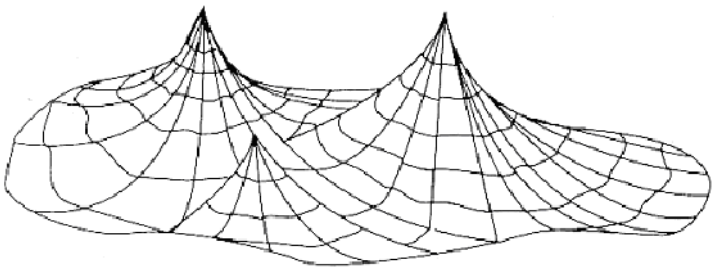

Figure 6.3: The fishnet organization [37]

6.3.2 Process and Project Oriented Approaches and the Hypertext Organization

Process and project orientation put organizations into a different perspective. They approach an organization as a system of processes instead of departments and hierarchy, and they analyze a series of ventures or projects instead of continuous business operations respectively [8]. In the process approach[4] grouping is performed by simultaneously applying all the principles that evolved from classical management theory, which means that work broken into pieces by scientific management is being reintegrated [15, p. 80]. The project approach to organizing subsumes task or project orientation as well as interdisciplinary team work. Projects are always time limited, so a project organization is time limited from the beginning of the project until its end [28, p. 44].

At a first sight, it seems impossible to combine these two approaches together[5], but their mutual benefits, with elimination of their disadvantages can be useful in the hypertext organization which was introduced by Ikujiro Nonaka [69, pp. 166–167]. This kind of organization consists of three layers – a business layer, which in essence is performing everyday bureaucratic tasks; a project team layer used for executing the multidisciplinary activities which increase the total knowledge of an organization; and a knowledge based layer that is imaginary and in which the knowledge accumulated in the previous two layers is once again categorized and put into new contexts [105, pp. 165–168]. Figure 6.4. shows a short outline of the hypertext organization.

[4]Which is very important due to the business process re-engineering and similar paradigms that enable business system management through business processes and their support through information technology.

[5]The process based approach is oriented to everyday operations, while the project based approach is oriented to certain ventures which generally are not repeatable and are time limited.

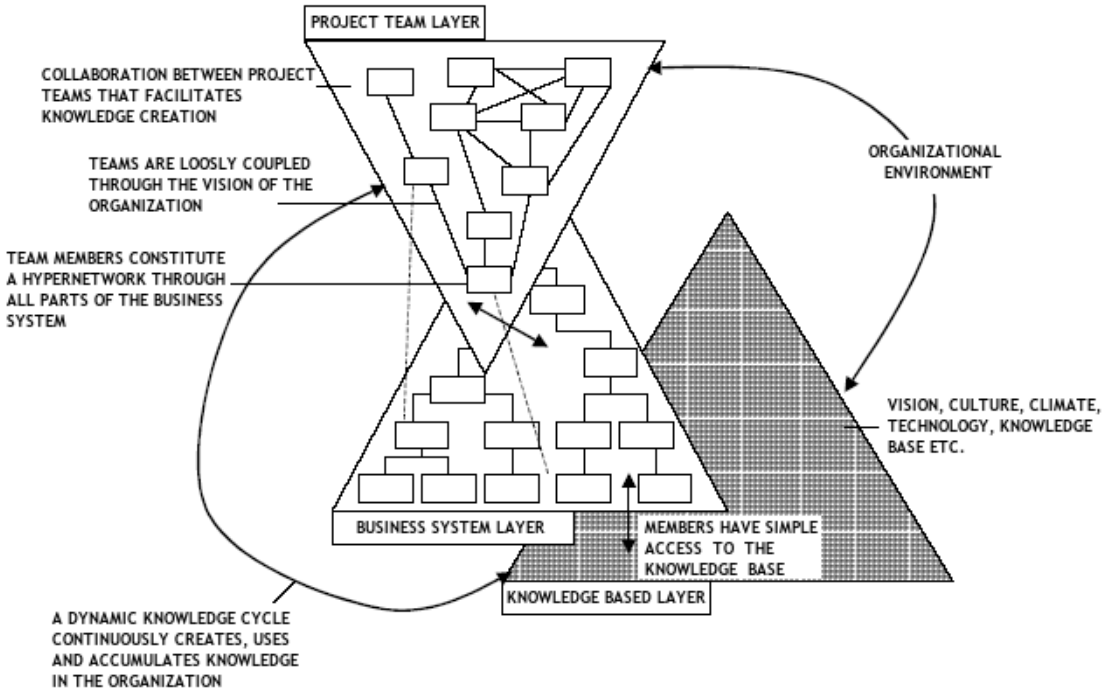

Figure 6.4: The hypertext organization [69]

6.3.3 Organizational Suprastructures and the Virtual Organization

Ad hoc suprastructures are concepts that are built on top of existing organizational structures and they emerge as a response to some problem or change in the immediate environment of the organization [105, p. 119]. Ad hoc organizations are characterized by adaptability, readiness, individual initiative, desire for experimentation, creativity, and outside growth and support [5, p. 7]. They usually disappear when the environment problem is solved.

A Virtual organization[6] is a target oriented suprastructure of geographically separated entities (organizational units) that are specialized for a predefined area of activity, are interconnected through space, time and organizational limitations, mostly using information, communication and network technology for efficient and flexible cooperation and exchange of knowledge. Figure 6.5. shows the concept of a virtual organization [7].

[6]Virtual organization is one of the most widespread examples of ad hoc organization in expert literature. [7] says that these organizations exist in cyberspace, that they develop proportionally with the development of information and communication technology and that they can be found in conventional organization structures. Under the term cyberspace he understands the media in which electronic communication and computer programs exist, and he argues that the understanding of the term is essential to the understanding of the virtual organization.

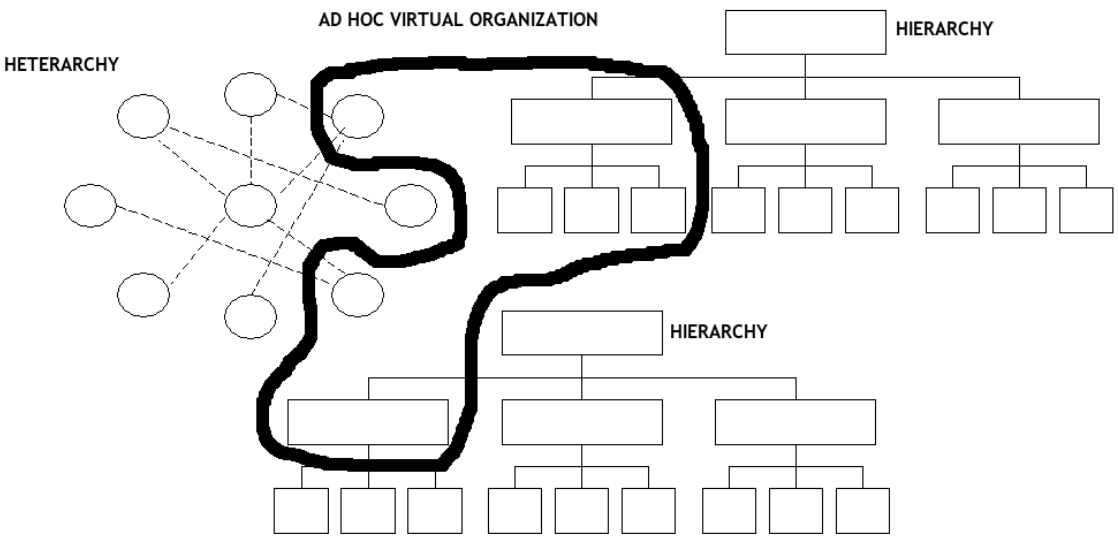

Figure 6.5: The virtual organization [7]

6.3.4 Organizational Architecture

At more recent times some authors introduce the concept of organizational architecture that does not only include the formal organization, but also the informal, the business processes, business strategy as well as human resources as the most important factor of the organization [68, p. 4]. It seems obvious that the metaphor from conventional architecture implies a connection of organizational structure with other systems within the organization into a unique synergistic system that will achieve more than just the sum of its parts [63, p. 2]. Figure 6. outlines the basic concepts of organizational architecture.

The objective of organizational architecture is to develop an organization that will be able to continuously create new values for its customers as well as to organize and optimize it self [91]. Other authors understand under organizational architecture building blocks that are necessary for organizational growth like organizational structure, organizational culture and human resource development. Thus organizations have to learn how to design, implement and manage these blocks [20, p. 1].

Organizational architecture is closely bound to organizational design and thus Nadler and Tushman define it as a wide set of decisions that have to be made by managers in their organizations. Since under organizational design only a part of this set of decisions is understood they decided to name this wider set organizational architecture [68, p. 4].

As one can conclude from the previous reasoning different authors consider

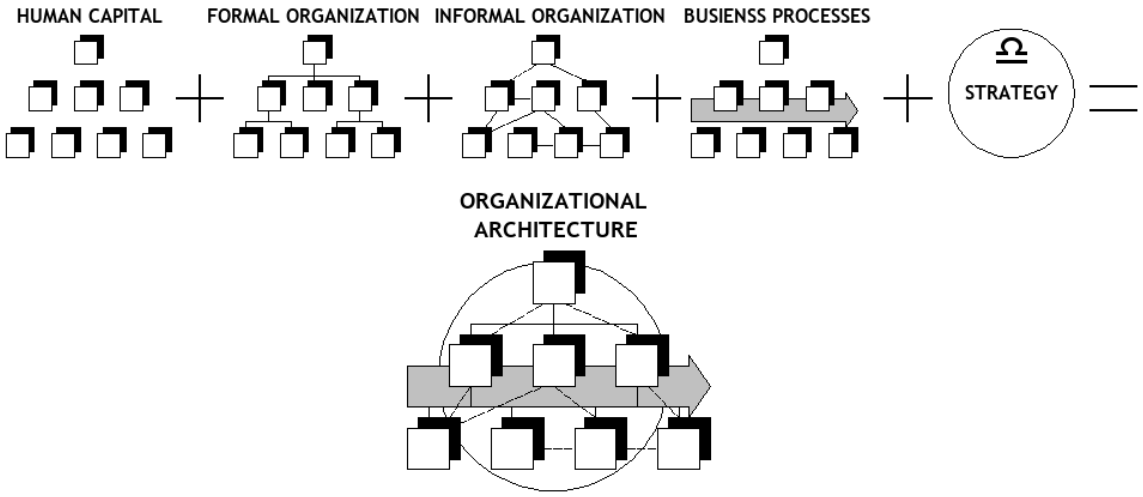

Figure 6.6: Basic concepts of organizational architecture [105]

different components of organizational architecture [63, 32, 20, 21], but one can recognize the five most important components of organizational architecture: the formal organization (organizational structure), the informal organization (organizational culture), the business processes, strategy and human resources [107, p. 41] as shown on figure 6.6 [105, p. 2].

6.3.5 The Fractal Company

The concept of a fractal company (Ger. die Fraktale Fabrik) was first introduced by Hans – Jürgen Warnecke in 1992. who has concluded that organizations are similar to complex systems that are characterized by fractals [111]. This concept was in a way an answer to similar Japanese and American concepts adapted to the European market [1, p. 1].

The term fractal was introduced by Mandelbrot to denote an object that has a certain degree of statistical self-similarity on every observed resolution and is generated by a infinite number of recursive iterations. If one observes a fractal (figure 6.7a.) she can recognize a certain pattern. By taking a closer look (possibly under a magnifier) she can observe the same pattern on lower and lower levels.

As one can see on figure 6.7b. a fern twig has some characteristics of a fractal (one twig is similar to the smaller twigs it consists of, which in turn consist of even smaller twigs). If one applies this concept to organizational structure she can observe fractals in the form of individuals, departments, divisions, process flows, decisions and all the other systems that make up an organization. The main objective is to find the fundamental pattern that will yield deeper insight to the organization as a whole [105, p. 150].

A fractal in Warnecke's sense is an autonomous organizational unit that has

objectives and a function that can be clearly described. Typical characteristics of a fractal are self-similarity, self-organization and self-optimization [1, p. 1].

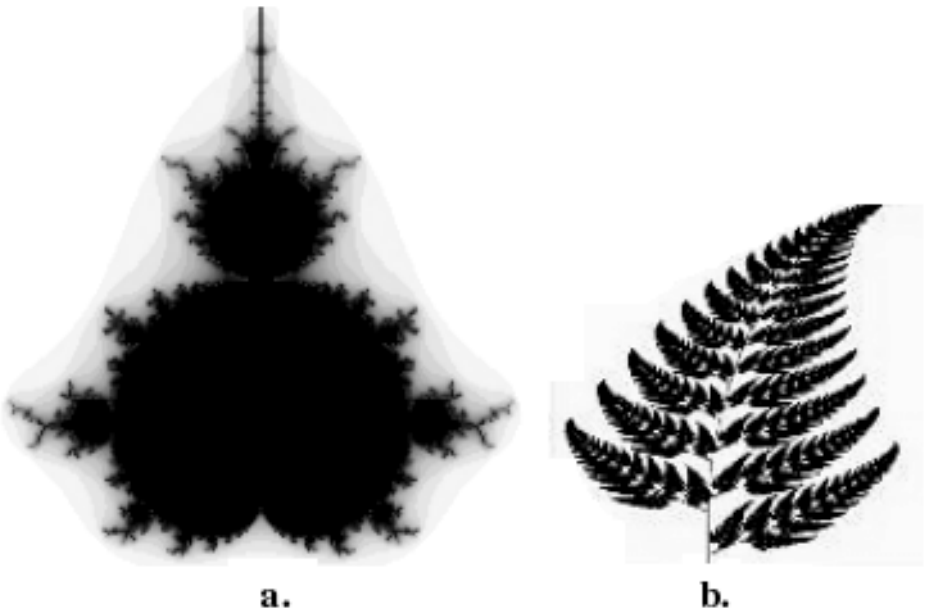

Figure 6.7: a. The Mandelbrot fractal, b. A fern twig [105]

Self-similarity means that the goals of particular fractals (from the individual in the organization, until the organization as a whole) match into a harmonic mutual objective. Self-organization means that particular fractals have their own autonomy concerning ventures and decisions according to the self-similarity rule, e.g. objectives have to be harmonized with upper and lower fractals. Self-optimization means that fractals continuously optimize their self-initialized work and decision making [76, p. 34]. Figure 6.8. shows the fractal principle where the spiral connecting the individual fractals represents the business process.

6.4 Relations between Social, Organizational and Information Systems

Brumec developed a genetic as opposed to descriptive definitions of information systems as follows: "*An information system is a subsystem of the organizational system, whose task is to link processes on the operational, management and decision-making level. Its goal is improving performance efficiency, supporting good quality management and increasing decision-making reliability.*" [13]. An information system comprises of information and decision flows between these organizational processes as shown on figure 6.9

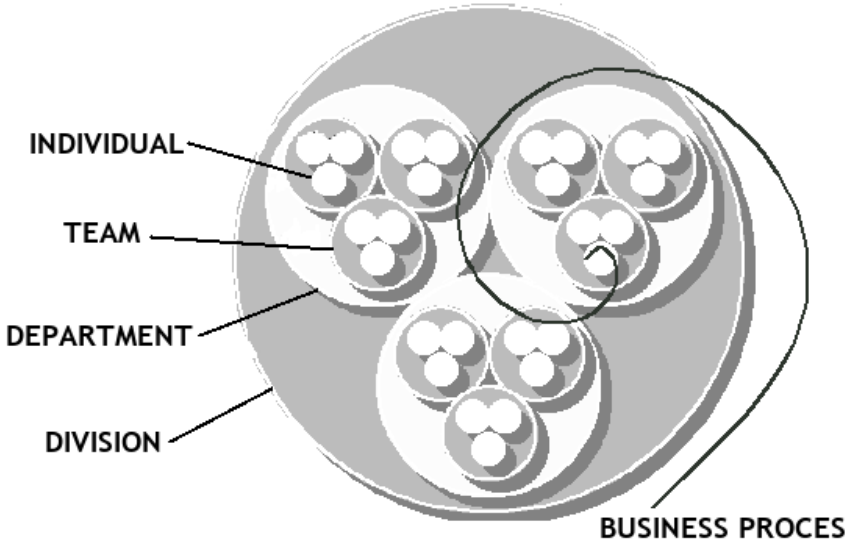

Figure 6.8: The fractal principle [105]

The consequence of such a definition is that an information system cannot exist by itself. It is always a subsystem of some real organizational system, i.e. each organizational system has its unique and distinctive information system. An information system can but doesn't have to be supported by information and communication technologies (ICT). These relationships are depicted on figure 6.10 which is an adaptation of [44].

More recently due to the development of the Internet and especially so called Web 2.0 and Web 3.0 applications one was able to observe systems supporting information flows inside social systems. We could easily call this kind of systems social information systems since they comprise the same elements as information systems defined by Brumec except that they are subsystems of a larger class of systems than organizations. They are subsystems of social systems comprising of their information flow and used to facilitate social functions and decision making.

Thus we can conclude with the flowing set of relations: organizations as well as social information systems are subsystems of social systems. Information systems (in Brumec's sense) are subsystems of organizations. Information systems as well as social information systems can be but do not have to be supported by ICT.

6.5 A Critical Review of Autopoiesis

Prior to an attempt to define autopoiesis in the context of information systems we need to clarify our view on autopoietic theory. In terms of Maturana and Varela autopoiesis consists basically of two parts: (1) preservation of organization and (2) regeneration of

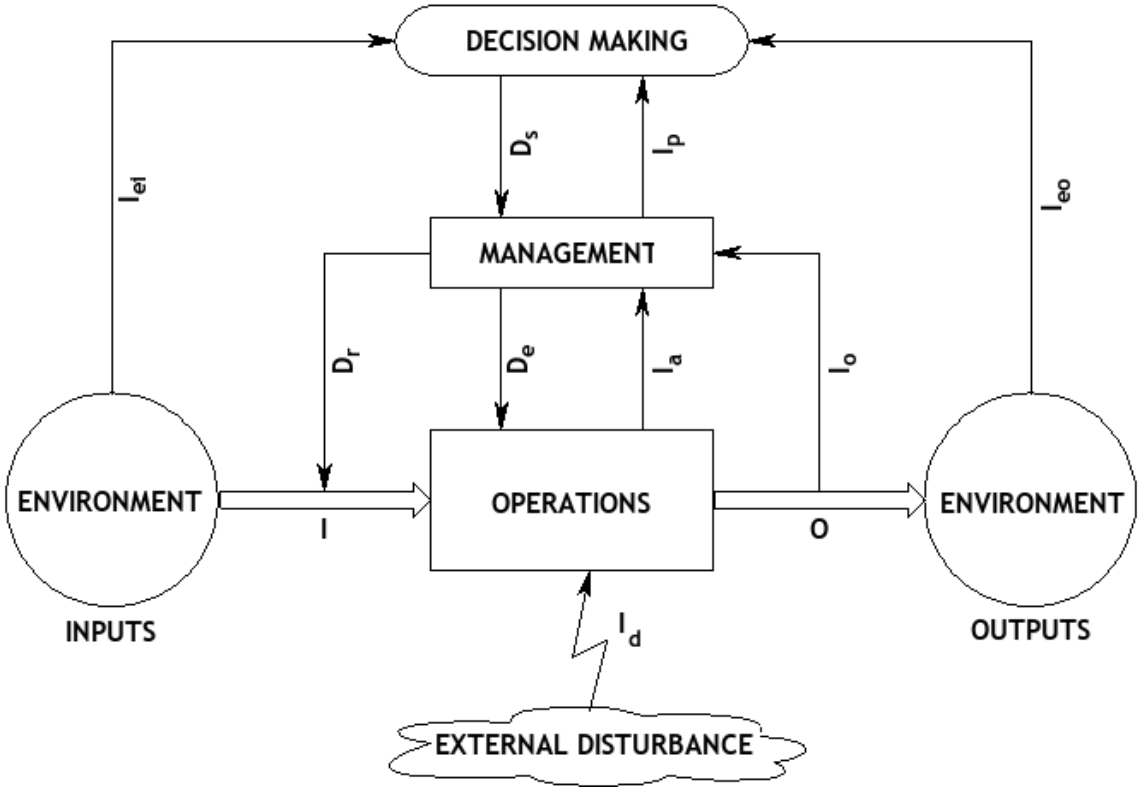

Figure 6.9: The information system as a subsystem of an organization [13]

structure. While the letter seems to be obvious the former raises questions outlined in some critics of the theory. We shall try to depict these questions using some simple examples.

Let us observe a living being that naturally changes due to metamorphosis for a most impressive example. "*A caterpillar organization auto-organizes to a larger caterpillar organization or pupa organization, and pupa organization in a butterfly organization.*" [99] If we follow Maturana's and Varela's reasoning in this case we would have three distinct autopoietic systems: a caterpillar system, a pupa system and a butterfly system since processes in these systems are in different relations even if we are talking about one and the same entity. We can extend this example to any living beings since living beings are born, evolve, eventually reproduce themselves, age and eventually die.

Now let us observe a living being that changes drastically due to environmental influences. For example a cat looses its tail due to an accident. The system isn't able to maintain its organization (when following Maturana and Varela) since part of the structure (particularly components that were part of the cat's tail with accompanied resources) are gone. The organization would be preserved if the components could be regenerated, but nature tells us that cats do not regenerate their tails once loosing it. Relations between processes that were performed in the cats tail are gone. But, part of the organization is

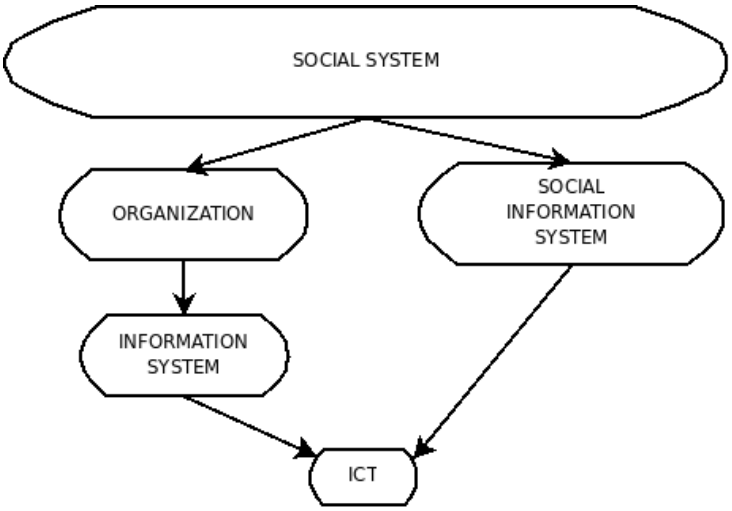

Figure 6.10: Relationships between the social system, organizations, information systems and ICT (Adapted partially from [44])

still maintained, as well as autopoiesis since the cat is still a living being.

Another interesting observation would be the one of a vine tree. It is well known that one can cut a twig of a wine tree put it into soil and under certain circumstances the twig will root and become a tree of its own. In terms of Maturana and Varela the organization of the initial wine tree was split into two distinct parts and both of them became a system of their own. But which of them is the original one, if any? We can extend this example to any reproducing species, and ask the famous question when does life and consequently when does autopoiesis occur?

All these examples let us seriously consider that organization (in Maturana's and Varela's sense) has to change during time. Even if they describe autopoietic systems as processes they do not seem to include basic system dynamics like evolving, aging or metamorphosis as a visual example. So we consider that organization can change but in a natural (evolutive) way, only if certain preconditions are fulfilled. Organization evolves, matures, eventually reproduces itself, ages and dies. From this point of view we can depict the most basic (evolving) autopoietic system as shown on figure 6.11 whereby P1, P2, P3, ... , Pn are instances of the same process performed in different time frames, whilst R1 is a single component holding the resources needed for the processes to perform. The component will change during time, as well as the organization comprised of the relations between process' instances by evolving in a natural way.

The example of the cat let us consider that not the whole organization has to be maintained for a system to maintain autopoiesis. So we introduce the notion of a core part of organization – the systems identity. A system will remain autopoiesis if its

identity remains. This notion of identity lets us also explain the example of the wine twig and reproduction of living beings. In the former only a part of the organization (that isn't in the identity of the system) was split of and produced an identity of its own. The original tree remained its identity and thus maintained autopoiesis. In the letter case a living being will create a new part of its own organization that will eventually yield a new identity of a new living being. This process is usually denoted by reproduction.

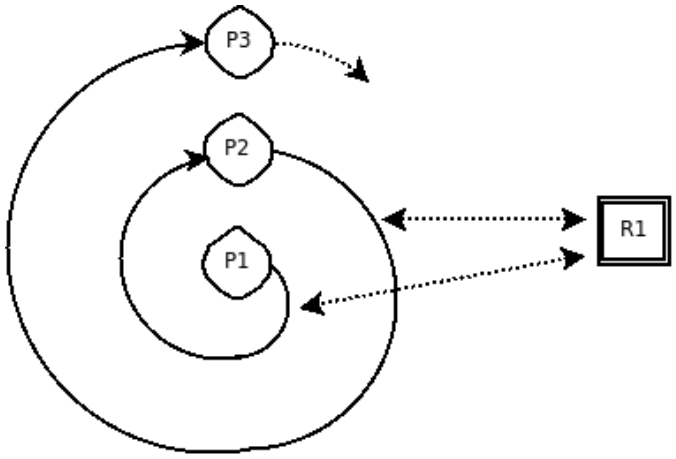

Figure 6.11: The basic (evolving) autopoietic system

In terms of Luhmann social systems are sense processing systems of communication. They are autopoietic in terms of reproducing communication. In the sense of Maturana and Varela structure would be comprised of communicative events that are reproduced. Organization would be the system of relations between these communicative events. In the original sense of Maturana and Varela this organization had to be preserved. But, relations between communicative events change due to the evolution of social systems. Thus, organization, as we reasoned previously, evolves and changes in a natural way.

If we take the global social system (society) as an example we could ask the question if this system still existed after catastrophes like the tsunami or the civil war in Rwanda? Even if a great deal of stored communication in all these victims minds disappeared the system still remained since its identity survived.

Another questionable statement is that social systems are systems of communication and only of communication. Especially societal and organizational systems but in a way interactional systems as well rely on stored communication. The question is where is this communication stored and can we conceptualize autopoiesis of these systems in another way that would yield a better understanding of social systems.

To answer this question let us take a most simple example of a flock. We

could say that relationships between the processes conducted by the animals in the flock comprise an organization that emerges through communication and perception. These relationships set up certain roles during time inside the flock (like the alpha male). We could furthermore define a structure to be comprised of the animals (components) of the flock as well as their characteristics in the perception of the others. It is important to state here that the components of the structure are not any animals but animals accepted by the other animals, that in turn accepted to be part of the flock. Thus structure changes during time but the organization remains, and this simple system of a flock could be considered an autopoietic system.

If we take this example into a social systems' perspective we can conceptualize social systems as systems comprising of accepted individuals that accepted to be part of the social system. These accepted individuals are reproduced (their acceptance, their social roles, expected attitudes and manners, their beliefs but not the individuals themselves) and thus comprise the components of the social system that build up the structure. The organization (in Maturana's and Varela's sense) is comprised of the relations between the accepted individuals that are build up through social processes of communication. Organization remains constant but evolves in a natural manner. This reasoning introduces individuals as a new idea into Luhmann's social system perspective that we missed in some extent. These individuals are exchanged during time and reproduced (not physically but socially).

6.6 Defining Autopoietic Information Systems

Having the previous reasoning in mind we could define the following classes of autopoietic systems: (1) biological systems – autopoietic systems in the sense of Maturana and Varela with the addition of organization's identity and dynamics, (2) social groups of biological systems (flocks, swarms etc.) - systems that are comprised of relationships between living beings whereby during time roles evolve which are attractors that lead future development of the system, (3) social systems – special cases of social groups where biological systems are mostly humans, (4) information systems – subsystems of social systems that deal only with information and communication inside them. Social systems can further be divided into: (a) interactional, (b) societal, (c) organizational having corresponding information systems.

Thus, autopoietic information systems would be interactional-, societal- and

organizational- social systems in the sense of Luhmann since they deal exclusively with communication. Since social systems from our perspective are systems of accepted individuals we can state according to the genetic definition of information systems that their subsystems dealing with information are their respective information systems.

An autopoietic information system is then defined as a set of relations between communicative events that reproduce new communicative events based on previous (stored) communication. The organization of this system (in Maturana's and Varela's sense) are the relations between communicative events described through their semantics (meaning). The structure of the system (in Maturana's and Varela's sense) are the means that are used to produce communication described through syntax.[7]

Interactional autopoietic information systems are systems that emerge and do virtually not depend on previously stored communication but on current interactions between communicative events. Interactional autopoietic eventually yield societal autopoietic information systems when attractors of meaning emerge that are reproduced through stored communication. A special case of societal autopoietic information systems are organizational autopoietic information systems that primarily consist of decisions that set up the possible future states of the system.

To approach the previously defined concepts of modern organizations we need provide a suitable framework for description of these concepts. In the following we shall use the terms structure and organization as well as identity in the (extended) sense of Maturana and Varela (if not stated otherwise).

A heterarchy in terms of autopoietic theory can be defined as an organization in which relations aren't strictly defined but rather activated due to some changes in the structure. Since the structure is comprised of accepted individuals (the components) thus any influence from the environment is detected in the structure. The components activate relations by making decisions about the situation in their immediate environment. Thus the information system of an heterarchy would be comprised of decisions made on behalf of indirect influences from the environment that activate relations inside the heterarchy.

A fishnet organization, as a special case of heterarchy activates special relations between the components of the system. This special relations when activated by an adequate number of components create a role (the top of the dynamic hierarchy, e.g. managerial role, project leader etc.). These roles disappear when the relations are dis-

[7]To see an alternative definition of autopoietic information systems using an descriptive rather than a genetic approach take a look at [8].

activated by the components of the system. Thus the information system of a fishnet organization is comprised of decisions that create special types of roles.

The hypertext organization in these terms may be defined as an organization of partially relatively static and partially state dependent relations. The relatively static part of relations comprises the bureaucratic (process oriented) part of the organization. The state dependent part is activated when a project is started. Thus the information system of the hypertext organization is comprised of decisions that activate projects and increase the knowledge in the organization's knowledge base.

Ad-hoc suprastructures can be defined as organizations comprised of emergent relations. These relations emerge due to indirect influences from the environment or due to direct changes in multiple structures. As soon as these influences disappear so do the relations as well as the ad-hoc suprastructure. The information system of a suprastructure thus consists of decisions that create new relations between existent components of different structures.

The virtual organization is a special case of ad-hoc suprastructure in which relations are activated through decisions in multiple structures that connect distributed components through cyberspace. The only difference between ad-hoc suprastructures and the virtual organization is that the letter exists in cyberspace. The information systems are equivalent except that the virtual organization's information system is fully supported through information and communication technology.

Organizational architecture is considered to be a framework for a holistic approach to any organization (in the institutional sense). Human resources comprise the components of the organization. The formal and the informal organization comprise two distinct sets of relations building up a single organization. Business processes define the interactions between the components (the structure) and the environment. The information system of any organization (institutional sense) is thus the system of decisions that build up the organizational architecture. Strategy is an important subsystem of the information system that coordinates the system's organization.

The notion of a fractal organization can yield deeper insight to this holistic view. Subsystems of the organization (fractals) are self-similar. This means that decisions will also be self-similar which in turn means that there are certain patterns in the information system that are self-similar and can be observed. By taking advantage of this fact one could develop new approaches to strategic planning.

Following these arguments we can conclude that to support autopoiesis in

modern organization's information systems we need to support the following types of decisions: (1) decisions made by components on behalf of influences from the environment, (2) decisions that create relations which build up managerial roles, (3) decisions that activate projects, (4) decisions that create new knowledge, (5) decisions that create new relations between different structures, and (6) decisions that coordinate the system (strategy). Other important concepts that should be supported are: (1) interactions between the system and the environment, (2) management and recognition of patterns inside the system, (3) filtering of complexity and (4) boundary determination.

6.7 Modern Information and Communication Technologies

In order to support an autopoietic information system by technology we give a brief outline of modern information and communication technologies that emerged due to the amazing growth of the world's major network. These technologies are often referred to with the term Web 2.0 or more recently Web 3.0 even if it sometimes isn't clear what this term subsumes. In a general perspective we can say that Web 2.0 subsumes user participation through communication and content creation, whilst Web 3.0 tries to incorporate ideas from the semantic web into a social web perspective.

Forum A forum is a network application that allows its users multimedia based communication (mostly through text, images, and simple animations) and is organized into subjects and sub-forums. A forum is hierarchically organized in a way that every user can participate in the communication process by answering previous messages. The communication process of such a system can be thought of as a general tree structure in which nodes are messages and arcs are the essential connections between message and answer. Forums are a very widespread technology with a lot of implementations like PHPbb and vBuletin to name the most popular as well as lots of communities functioning almost completely through this kind of technology.

Wiki The concept of a wiki system operates in the following way: every user or visitor of a wiki service on the Web can change articles and information that he encounters, add new articles and/or information and argue about the existing ones. An additional mechanism that is built into such systems is the possibility to interconnect terms used in articles. In other words, every term that is mentioned in one article of

the system can be connected with other articles which elaborate it further. This mechanism allows easier explanation of unknown terms to users. A disadvantage of such a system is the lack of a mechanism for consistent decision making. In other words, such systems are sometimes affected by so called editor wars, when users fight each other by constantly changing some disputable article content. Wikis became as well as forums a widespread technology with typical implementations like WikiMedia to name the most prominent one. WikiMedia is the engine of Wikipedia the free Internet encyclopedia that was established in 2001. As of the time of writing this chapter Wikipedia had over 2.2 million articles in its English version, while there are versions for almost any world language.

Social networking Applications for social networking allow their users to virtually create social networks of their friends, colleagues, co-workers etc. One can browse others friend lists and profiles, play virtual games, get in touch with long lost friends etc. Some of the most famous social applications like Facebook or MySpace allows one to engage a lot of different activities with her friends like games, projects, petitions, causes, exchange images, videos, journals etc.

Social tagging Social bookmarking and social tagging technologies allow their users to organize content they encounter on the web or on site through tags and/or bookmarks. One of the most prominent social bookmarking and web search engine application del.icio.us allows users to tag any page on the World Wide Web with custom defined keywords. The search results are impressive having the simplicity of the algorithm that constitutes the application in mind as opposed to complex algorithms used by traditional search engines.

Content feeds Content feeds or web syndication is a popular technology in which a section of a website is made available for other sites and applications to use. This serious of protocols and standards allow users to aggregate information form different sources in one place.

Pod-casting Pod-casting services are another interesting web technology that allows its users to broadcast their own video material. Services like YouTube, Google Video and others became extremely popular and are often compatible with other technologies mentioned previously. Web services are modern networking technologies that enable remote procedures or services usage as if they were local. They enable the

development of distributed networking applications without the need to contain all the parts of it on a single computer or server. Extensible markup language (XML) is a data description language used in such services and content feeds. It is very simple and intuitive and is often used in conjunction with web services for interchange of data between the local application and the service. Business to business (B2B) communication is a modern communication concept between different organizations, distinguishing it from Business to Customer (B2C) communication model used by the organizations to communicate with their customers. B2B often relies on concepts such as web services and XML for the interchange of data.

P2P Peer-to-Peer is a group of network protocols which, instead of a usual client-server model, enable every participant to simultaneously be both client and server. The concept is based on mutuality in a way that every user shares certain contents at disposal to other users what gives him the right to access their shared content. P2P protocols aren't typically used for interpersonal communication, but for the exchange of electronic data.

Semantic web Web pages and the structure of the World Wide Web are adjusted to humans who are able to find, combine, internalize and reason about such stored knowledge. But, if someone tries to do the same using a computer program problems occur due to unstructured, distributed and semantically unadjusted sources of knowledge. Thus the semantic web is a systemic attempt of formalizing knowledge on the World Wide Web to facilitate more effective computer based retrieval and reasoning about knowledge. Semantic wiki systems are an additional idea to combine ideas from the semantic web with the dynamic and collaborative creation of content happening in wiki systems by adding meta information to created content. Semantic web services are another interesting idea of combining structured knowledge with web service to enable semantic retrieval, performing, connecting and interoperability of web services [62].

Open Source The approach of building information systems and applications based on open source is used by many very successful systems[8] like Apache, Perl, Wikipedia, Mozilla and Linux, as the most popular example. The concept of an open source

[8]Many successful systems, probably much more than open source systems, were developed in a closed source environment using traditional software engineering methodology.

project[9] functions in the following way: a programmer (or few of them) start an information system or application development project. All the source code they produce, the application and the documentation is public accessible, usually via Internet. Users play an important role in the system development process, they test it, check it, make suggestions, report bugs, criticize functionality etc. If the application or information system is widely used, it becomes more and more aligned with the customers' needs and its environment [105].

6.8 Current System Model

The development of the ᴛᴀOPīs [10] system was started in 2004 by a group of enthusiasts with the idea of creating a completely decentralized self-organizing project management system for use in public and political projects as well as in dynamic organizations.[11]

The project was managed in an Open Source manner so many ideas and concepts outlined here[12] are results of discussion thought various forums, mailing lists as well as wiki systems. A modified methodology of strategic planning of information systems [14] was adopted to create an initial model of the system as well as to plan future steps of the project. During time and due to Open Source development more and more ideas came into play and the initial model of the system was considerably changed.

In the following we will try to outline some basic concepts of the ᴛᴀOPīs system's architecture. The main structure of the system is a system similar to Open Source project management systems like SourceForge, RubyForge and others with a little extension in project semantics since any kind of project can be managed through the system not only information system projects. Any person or group of people can start their own project on the system and use it to manage the project in a completely distributed way. Any person can join any project and contribute to it.

Every project basically consists of a forum system for discussion as well as a wiki system for content creation. The forum system can be used for discussion between contemporary project members. Due to the fully decentralized nature and other ideas of self-organization there was need to eliminate the role of a forum moderator who is

[9] Not to be mistaken with classical (commercial) application development, where a development company in addition to the application sells the customer the source-code of the application.

[10] Initially ᴛᴀOPīs meant TiAktiv Open Politics Information System but more recently the acronym was renamed to Transparent Open Public Autopoietic Information System.

[11] A history of the ᴛᴀOPīs system as well as the various people, organizations and on-line communities is outlined in [85]

[12] Under which the idea of using autopoiesis as the main paradigm outlined by prof. A. Lauc

basically a privileged user that can filter, delete and/or change content on the system. Thus a filtering system was developed that allows every project member to be a moderator if she chooses to or to use the moderation of some other member. A list of moderators is provided that lets members choose the most popular moderators.

Another important issue was how to determine project leaders that will make decisions and use strategy to continuously improve project performance. People's opinions about other people change during time, thus as in the case of moderators project leader had to be a dynamic role that could be changed depending on people's opinions. An interesting idea was to use a modified PageRank algorithm that is used by the famous search engine Google to rank pages, in order to determine project leaders dynamically. The PageRank algorithm has its roots n social network analysis since it uses incoming and outgoing links of web pages to determine a page's rank. Every incoming link (e.g. another page that is linking to the page under consideration) is considered to be a vote for the page under consideration, while every outgoing link (e.g. links pointing to other pages on the web) are considered to be votes for the pages that they link to. Votes (links) are weighted with the rank of every page and the sum of all pages' ranks in a network is 1.

If we apply this idea to a social network in which people vote for zero or more others we can dynamically establish a hierarchy due to people's ranks. This algorithm was implemented in the ΤΑΟΡΙΣ system to determine member ranks as well as to determine the project leader.

An interesting analogy is the previously mentioned fishnet organization that is implemented in this way [88]. Since there are multiple projects active on the system there are multiple dynamic hierarchies that change during time depending on people's opinions.

To completely decentralize any generic function of the system additional projects were defined dealing with administration, development, support and Spam filtering. Thus administration of the system is a project where members administer the system and administrators are defined through the dynamic PageRank algorithm. Likewise all the other projects deal with the previously mentioned generic information system functions.

To employ people's knowledge and creativity in an nonobligatory way a tagging system was developed that allows every user to organize content she encounters on the system using attribute-value tags. These tags are meta information added to content that allow a possible intelligent agent to reason about knowledge on any project as well as on the system as a whole. Thus options that allow the export of a project's semantic web

ontology consisting of all the meta information users created by organizing content for themselves were implemented. The syntax of the wiki system was extended with a frame logic based language that allows users and computer programs to make dynamic queries and reason about knowledge stored in a projects ontology.

Another issue was how to combine this system with existing information systems as well as how to extend functionality (since the system provides only generic project and knowledge management functionality). Thus the idea of semantic web services as well as script extensions came to attention. Using web services one could combine existing information systems with the ┬AOPīs system. This functionality is planned but not implemented fully. There are however bindings to other popular communication systems like mailing lists, content feeds, social bookmarking as well as pod-casting services. Other planned features include social networking facilities to allow users to get in touch with their friends, colleagues, co-workers etc. but also to establish semantic relations between projects as well as to implement a simple peer to peer system to connect various instances of the ┬AOPīs system.

From an autopoietic theory perspective we can say that we were able to implement support for decisions that activate projects (since every user when encountering some opportunity in the environment can start a project), decisions that set up managerial roles (since every project member can vote for other members and thus decide upon project leadership), and decisions that create new knowledge (since every project member can create content on the semantic wiki system as well as tag any content on the system to organize knowledge). Decisions made by components on behalf of influences from the environment, decisions that create new relations between different structures as well as decisions that coordinate the system are only partially supported but not strictly formalized as the previous ones. Thus mechanisms to support these types of decisions have to be developed in the future.

On the other hand we were able to support interactions between the system and its environment (due to various connections to other systems as well as through the planned semantic web services and script extension's feature), filtering of complexity (through the filtering system) as well as boundary determination (only decisions stored on the system are part of the information system). The concept of management and recognition of patterns inside the system is not supported yet, but we plan to develop a system that will be able to set up relations between different projects and organizations as well as to recognize patterns and fractals inside the system and make use of them in a

semantic web perspective.

6.9 Experiences and Lessons Learned

The most important lesson we learned during the implementation of the ᴛᴀOPīs system is that autopoiesis is something that happens not something that can be implemented. Nevertheless dynamic organizations can have benefits if taking concepts from autopoietic theory into consideration when planning, modeling and developing their information systems. Through careful consideration autopoiesis can be facilitated in such systems.

By conducting an experiment with 160 students in a knowledge management course who were assigned to use the ᴛᴀOPīs system to acquire knowledge about a particular topic we gathered interesting insights [57]. Students were randomly divided in teams of 4 – 7 members with every team having a special topic that represented their project. After four weeks of cooperation results were impressive.

Since students were forced to cooperate with people they sometimes didn't even know the first week was quiet within a search for a leader. Students were told to use the ranking mechanism to find a leader who will communicate with "upper management" (the teachers). As soon as such a role was established work was divided into parts and teams started to conduct research on the topics.

Three weeks later impressive knowledge bases on the particular topics emerged consisting of lots of text encountered in different books, articles, and web sites. The semantic wiki systems were crowded with text, images, animations, short movie tutorials, meta information and queries that summarized information and put it into new perspectives. Still there were teams that weren't able to find a leader and such teams failed in the task to create a satisfactory solution.

After the projects were finished a survey was conducted to identify which criteria students used in establishing a leader role as well as how successful they would rate their projects. It is interesting that teams that used leadership skills as a criteria were able to identify a leader and were thus successful. On the other hand teams that didn't, weren't able to identify a leader and were less successful. Still on an average scale 82 % of the students rated their project successful and 84 % of students thought that their project leaders have leadership skills. If we take that students were divided into teams randomly which yields possible incompatibilities between students personalities into consideration these are impressive results.

Another lesson we learned is that to facilitate autopoiesis one needs to facilitate interaction. In the mentioned survey we also asked students for suggestions and improvements of the system and most of them answered that they want additional interaction systems (chat rooms, instant messaging, improved forum system, status of on-line members, collaboration). Other improvements that were suggested are improved user interface (better graphical user interface design, more user-friendly interface), additional functionality (better content formatting, additional query possibilities) and less system flaws.

6.10 Conclusion

In this chapter we developed a definition of autopoietic information systems based on the genetic definition of information systems as subsystems of organizations or social systems in a broader perspective. Following definitions of autopoiesis in biology, sociology and organization theory we proposed a definition of autopoietic information systems consisting of their semantics (organization in Maturana's and Varela's sense) and syntax (structure in Maturana's and Varela's sense). Syntax is exchangeable while the core part of semantics (the system's identity) remains stable. If the systems identity disappears so does the system. We argued that there are interactional, societal and organizational autopoietic information systems depending on the nature of communication or the reproduction of semantics.

By defining important concepts from modern organization theory through autopoietic theory we were able to identify decisions and concepts that have to be supported in order to support autopoiesis in information systems through information and communication technology. Modern Web 2.0 technologies that emphasize involvement seem to be a good platform for attending this goal.

Using these insights we developed such a system. During the development we learned that autopoiesis is something that happens not something that can be implemented as well as that autopoiesis happens through interaction. By supporting interaction one can support the emergence of autopoiesis in information systems.

To provide a formal backdrop for knowledge management in such systems we decided to use semantic wiki languages as described in chapter 5 with the addition of social network analysis metrics that can be used as a probability annotation.

Chapter 7

Programming Languages for Autopoiesis Facilitating Semantic Wiki Systems

In order to support autopoiesis in semantic wiki systems one needs to acknowledge the complex nature of the social system surrounding them. Complex systems are probabilistic in their very nature which is why we decided to annotate previously described semantic wiki languages with probability. This annotated probability value has to be a measure of truth, thus the main question is *how to characterize the probability that a certain person will say the truth?* The answer to this question can only be found in the laws of the social network the person participates in. Thus, we decided to use social network analysis to find this probability. In order to provide a suitable mechanism that will yield results based on peoples opinions about what is the truth, we shall implement an algorithm that will resemble the fishnet structure described above.

7.1 Social Network Analysis

In order to provide a suitable formal framework we shall familiarize ourselves with key concepts from social network analysis. Social network analysis is concerned with understanding the connections among social entities as well as with the implications of such linkages [112, pp. 17–20.].

Actor. The social entities which are under consideration are referred to as *actors*, and are discrete individual, organizational or collective social units. Examples of actors include

people in a group, organizational units within an organization, public service agencies within a country, countries within a international trade union etc.

Relational Tie. Actors are linked to each other by *social ties* like friendship, linking, respect, business transactions, lending or borrowing things, belonging to the same social club, talking together, exchanging e-mails, a road, river or bridge connecting two points, authority, kinship and many others.

Dyad. A *dyad* comprises of a pair of actors and the possible ties between them, whereby dyadic analyses focus on the properties of pairwise relationships. Such properties include reciprocity, whether specific types of relationships tend to occur together etc.

Triad Relationships can occur among more than two actors. A *triad* represents, for example, a subset of three actors and the eventual ties among them.

Subgroup. A *subgroup* is any set of actors including all ties between them.

Group. A *group* is a finite collection of all actors on which ties are to be measured. A system of ties consists of all the ties among a (more or less) bounded group.

Relation. A *relation* is a collection of ties of a specific kind between members of a group. Examples include friendship among children in a village, formal diplomatic ties among nations etc.

Having the basic terms defined, we are now able to define the notion of social network.

Social network. A *social network* is comprised of a finite set or sets of and the relation or relations defined on them.

7.1.1 Graph Theory

A more formal approach to defining social networks is graph theory [27, 112].[1]

Definition A *graph* $\mathcal{G}$ is the pair $(\mathcal{N}, \mathcal{E})$ whereby $\mathcal{N}$ represents the set of *verticles* or *nodes*, and $E \subseteq N \times N$ the set of *edges* connecting pairs from $\mathcal{N}$.

A graph can be represented with the so called adjacency matrix.

[1] There are off course other approaches like sociometrics.

Definition Let $\mathcal{G}$ be a graph defined with the set of nodes $\{n_1, n_2, ..., n_m\}$ and edges $\{e_1, e_2, ..., e_l\}$. For every i, j ($1 \leqslant i \leqslant m$ and $1 \leqslant j \leqslant m$) we define

$$a_{ij} = \begin{cases} 1, & \text{if there is an edge between nodes } n_i \text{ and } n_j \\ 0, & \text{otherwise} \end{cases}$$

Matrix $A = [a_{ij}]$ is then the adjacency matrix of graph $\mathcal{G}$. The matrix i symmetric since if there is an edge between nodes n_i and n_j then clearly there is also an edge between n_j and n_i. Thus $A = [a_{ij}] = [a_{ji}]$.

The notion of directed- and valued directed graphs is of special importance to our study.

Definition A *directed graph* or *digraph* $\mathcal{G}$ is the pair $(\mathcal{N}, \mathcal{A})$, whereby $\mathcal{N}$ represents the set of *nodes*, and $\mathcal{A} \subseteq \mathcal{N} \times \mathcal{N}$ the set of ordered pairs of elements from $\mathcal{N}$ that represent the set of graph *arcs*.

Definition A *valued* or *weighted* digraph $\mathcal{G}_\mathcal{V}$ is the triple $(\mathcal{N}, \mathcal{A}, \mathcal{V})$ whereby $\mathcal{N}$ represents the set of *nodes* or *verticles*, $\mathcal{A} \subseteq \mathcal{N} \times \mathcal{N}$ the set of ordered pairs of elements from $\mathcal{N}$ that represent the set of graph *arcs*, and $\mathcal{V} : \mathcal{N} \to \mathbb{R}$ a function that attaches values or weights to nodes.

A social network can be represented as a graph $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ where $\mathcal{N}$ denotes the set of actors, and $\mathcal{A}$ denotes the set of relations between them [64]. If the relations are directed (e.g. support, influence, message sending etc.) we can conceptualize a social network as a directed graph. If the relations additionally can be measured in a numerical way, social networks can be represented as valued digraphs.

One of the main applications of graph theory to social network analysis is the identification of "most important" actors inside a social network. There a lots of different methods and algorithms that allow us to calculate the importance, prominence, degree, closeness, betweenness, information, differential status or rank of an actor.[2]. Herein we would like to outline one of such metrics introduced by Bonacich [11] called eigenvector centrality, but for our purpose any other metric can be used that can yield an approximation of the probability that a certain person will say the truth in a meta data statement.

We believe that the knowledge is justified, true belief [69]. Thus our conceptualization of meta data statements as units of formalized knowledge will follow this

[2]See [112] for an in depth discussion of such metrics.

definition, making the probability of giving a true statement a matter of justification. A person is justified if other members of a social system believe in his statements. Bonacich takes a similar approach, and gives a metric that calculates the centrality of a node based on the centrality's of its adjacent nodes. Eigenvector centrality assigns relative values to all nodes of a social network based on the principle that connections to nodes with high values contribute more to the value of the node in question than equal connections to nodes with low values.

Definition Let p_i denote the value or weight of node n_i, let $[a_{ij}]$ be the adjacency matrix of the network. For node n_i let the centrality value be proportional to the sum of all values of nodes which are connected to it. Hence:

$$p_i = \frac{1}{\lambda} \cdot \sum_{j \in M(j)} p_j = \frac{1}{\lambda} \cdot \sum_{j=1}^{N} a_{ij} \cdot p_j$$

where $M(i)$ is the set of nodes that are connected to the ith node, N is the total number of nodes and λ is a constant. In vector notation this can be rewritten as

$$p = \frac{1}{\lambda} \cdot A \cdot p \text{ or as the eigenvector equation } A \cdot p = \lambda \cdot p$$

PageRank is a variant of the Eigenvector centrality measure, which we decided to use herein. PageRank was developed by the famous company Google, or more precise by Larry Page (from where the word play PageRank comes from) and Sergey Brin who were the founders of this company. They used this graph analysis algorithm, for the ranking of web pages on a web search engine. The algorithm uses not only the content of a web page but also the incoming and outgoing links. Incoming links are hyperlinks from other web pages pointing to the page under consideration, and outgoing links are hyperlinks to other pages to which the page under consideration points to.

PageRank is iterative and starts with a random page following it's outgoing hyperlinks. It could be understood as a Markov process in which states are web pages, and transitions (which are all of equal probability) are the hyperlinks between them. The problem of pages which do not have any outgoing links, as well as the problem of loops is solved through a jump to a random page. To ensure fairness (because of a huge base of possible pages), a transition to a random page is added to every page which has the probability q and is in most cases 0.15. The equation which is used for rank calculation (which could be thought of like the probability that a random user will open this particular page) is as follows:

$$\text{PageRank}(p_i) = \frac{q}{N} + (1-q) \sum_{p_j \in M(p_i)} \frac{\text{PageRank}(p_j)}{L(p_j)}$$

Where $p_1, p_2, ..., p_N$ are nodes under consideration, $M(p_i)$ is the set of nodes pointing to p_i, $L(p_i)$ the number of arcs which come from node p_j, and N the number of all nodes [12, 72].

A very convenient feature of PageRank is that the sum of all ranks is 1. Thus, semantically, we can define the ranking value of persons (or actors in the social network) participating in a given wiki environment as *the probability that a person will say the truth in the perception of the others.* In the following we will use the ranking, obtained through such an algorithm in this sense.

7.2 Probability Annotation

As shown in chapter 5 there are basically two types of statements wiki users can make to provide meta data: (1) attribute-value tags and (2) hyperlinks. Provided the wiki system has the right ranking and voting facilities, every user has its associated rank on the system. Thus we can define the annotation scheme of autopoietic semantic wiki systems as follows:

Definition Let $S = \{s_1, s_2, ..., s_n\}$ a set of meta data statements, $A = \{a_1, a_2, ..., a_n\}$ a set of authors' rankings, and let $\rho : S \times A$ be a corresponding authorship relation. Then the **annotation** $\overline{=}_{\wedge}$ of the meta data statements is defined as follows:

$$s \overline{=}_{\wedge} a_\Sigma, a_\Sigma = \sum_{(a,s) \in \rho} a$$

An extension to such a probability annotation is the situation when meta data statements can have a negative valency. This happens when a particular user disagrees to a meta data statement of another user. Such an annotation would be defined as follows:

Definition Let $S = \{s_1, s_2, ..., s_n\}$ a set of signed meta data statements, $A = \{a_1, a_2, ..., a_n\}$ a set of authors' rankings, and let $\rho : S \times A$ be a corresponding authorship relation. Then the **annotation** $\overline{=}_{\wedge}$ of the meta data statements is defined as follows:

$$s \overline{=}_{\wedge} a_\Sigma, a_\Sigma = \begin{cases} \sum\limits_{(a,s) \in \rho} a - \sum\limits_{(a,-s) \in \rho} a & \text{if } \sum\limits_{(a,s) \in \rho} a > \sum\limits_{(a,-s) \in \rho} a \\ 0 \quad \text{if } \sum\limits_{(a,s) \in \rho} a \leqslant \sum\limits_{(a,-s) \in \rho} a \end{cases}$$

Such a definition is needed in order to avoid possible negative probability (the case when dissagreement is greater then improvement).

7.2.1 Query Execution

In a concrete system we need to provide a mechanism for query execution that will allow users to issue queries of the following form:

$$Q_p : F \,\overline{\overline{\wedge}}\, p.$$

Where F is any formula in frame logic and p a probability. The semantics of the query is:

does the formula F hold with probability p with regard to the knowledge base?

The solution of this problem is equivalent to finding the probabilities of all possible solutions of query F

$$Q : F$$

Definition Let $R_Q = \{r_1, r_2, ..., r_n\}$ be a set of solutions to query Q, then R_{Q_p} is a subset of R_Q consisting of those solutions from R_Q which probability is greater or equal to p and represents the set of solutions to query Q_p.

The probability of a solution $p(r_i)$ is obtained by a set of production rules:

Rule 1 *If r_i is a conjunction of two formulas r_{i1} and r_{i2} then $p(r_i) = p(r_{i1}) \cdot p(r_{i2})$*

Rule 2 *If r_i is a disjunction of two formulas r_{i1} and r_{i2} then $p(r_i) = p(r_{i1}) + p(r_{i2})$*

Rule 3 *If r_i is an F-molecule if the form $i[an \rightarrow av]$ then $p(r_i) = \min(p(an), p(av))$*

The implications of these three definitions are given in the following four theorems:

Theorem 7.2.1 *If r_i is an F-molecule of the form $i[an_1 \rightarrow av_1, ..., an_n \rightarrow av_n]$ then $p(r_i) = \prod_{i=1}^{n} \min(p(an_i), p(av_i))$*

Proof Since r_i in this case can be written as:

$$i[an_1 \rightarrow av_1] \wedge ... \wedge i[an_n \rightarrow av_n]$$

Due to rule 3 the probabilities of the components of this conjunction are $\min(p(an_1), p(av_1)), ..., \min(p(an_n), p(av_n))$

Due to rule 1 the probability of a conjunction is the product of the probabilities of its elements which yields $\prod_{i=1}^{n} \min(p(an_i), p(av_i))$ ∎

Theorem 7.2.2 *If r_i is an F-molecule of the form $i : c[an_1 \rightarrow av_1, ..., an_n \rightarrow av_n]$ then* $p(r_i) = p(i : c) \cdot \prod_{i=1}^{n} \min(p(an_i), p(av_i))$

Proof Since the given F-molecule can be written as

$$i : c \wedge i[an_1 -> av_1] \wedge ... \wedge i[an_n -> av_n]$$

the proof is analogous to the proof of theorem 7.2.1. ∎

Theorem 7.2.3 *If r_i is a statement of generalization of the form $c_1 :: c_2$, and if P is the set of all paths between c_1 and c_2 and if $\blacktriangleright$ is the relation of immediate generalization then*

$$p(r_i) = \sum_{pa \in P} \prod_{c_j \blacktriangleright c_i \in pa} p(c_j \blacktriangleright c_i)$$

Proof Since any class hierarchy can be presented as a directed graph its obvious that there has to be at least one path from c_1 to c_2. If the opposite were true the statement wouldn't hold and thus wouldn't be in the initial solution set.

For the statement $c_1 :: c_2$ to hold, at least one path statement of the form

$$pa_x = c_1 \blacktriangleright c_{x1} \wedge c_{x1} \blacktriangleright c_{x2} \wedge ... \wedge c_{xn} \blacktriangleright c_2$$

has to hold as well. This yields according to rule 1 that the probability of one paths would be:

$$p(pa_x) = \prod_{c_j \blacktriangleright c_i \in pa} p(c_j \blacktriangleright c_i)$$

Since there is a probability that there are multiple paths which are alternative possibilities of proving the same premise, it holds that:

$$pa_1 \vee pa_2 \vee ... \vee pa_m$$

Thus from rule 2 we get:

$$p(c_1 :: c_2) = \sum_{pa \in P} \prod_{c_j \blacktriangleright c_i \in pa} p(c_j \blacktriangleright c_i)$$

what we wanted to prove. ∎

Theorem 7.2.4 *If r_i is a statement of classification of the form $i : c$ then*

$$p(r_i) = p(i) \cdot \sum_{pa \in P} \prod_{c_j \blacktriangleright c_i \in pa} p(c_j \blacktriangleright c_i)$$

Proof Since the statement r_i can be written as:

$$r_i = i : c_1 \wedge c_1 :: c$$

the given probability is a consequence of rule 1 and theorem 7.2.3. ∎

7.2.2 Query Execution with User-Defined Rules

A special case of query execution is when the knowledge-base contains user-defined rules due to `rule : Head :- Body` tags. Such rules are also subject to probability annotation, since such an attribute-value tag is a valid meta data statement. Thus we have:

$$\text{rule} : Head \leftarrow Body \,\overline{\overline{\wedge}}\, p$$

where p is the annotated probability of the rule. In order to provide a mechanism to deal with such probability annotated rules, we will establish an extended definition of rules for semantic wiki languages:

Definition If some object o is tagged with attribute `rule` and the corresponding value is a valid frame logic rule of the form `Head :- Body` then this meta data statement is removed from object o and the following rule is added to the knowledge-base:

$$Head \leftarrow Body \wedge CounterPredicate$$

whereby $CounterPredicate$ is a predicate which will count the number of times the particular rule has been successfully executed for finding a given solution.

The query execution scheme has to be altered as well. Instead of finding only the solutions from formula F an additional variable for every rule in the knowledge-base is added to the formula. For n rules we would thus have:

$$Q : F \wedge count(?r_1) \wedge count(?r_2) \wedge \cdots \wedge count(?r_n)$$

In order to calculate the probability of a result obtained by using some probability annotated rule we establish the following definition:

Definition Let r be a result obtained with probability p_F by query F from a knowledge-base, let p_r be the probability of rule R, and c the number of times rule R was executed during the derivation of result r. The final probability of r is then defined as:

$$p(r) = p_F \cdot p_r^c$$

This definition is intuitive since for the obtainment of result r the rule R has to hold c times. Thus if a knowledge-base contains n rules $(R_1, \ldots, R_n)$ and their corresponding annotated probabilities are $p_{r1}, \ldots, p_{rn}$ and numbers of execution during derivation of result r are $c_1, \ldots, c_n$ then the final probability is defined as:

$$p(r) = p_F \cdot \prod_{i=1}^{n} {p_{ri}}^{c_i}$$

7.3 Annotated Semantic Wiki Language

To define annotated semantic wiki languages we need to extend the definition of queries to support annotation. This is done with the following simple production rule:

```
<query> ::= <query_start>
            <frame_logic_query>
            <probability_constraint>?
            <semantic_template>
            <query_end>
```

Additionally the definition of meta data statements has to be altered to take annotations into consideration:[3]

[3]For a complete implementation of niKlas grammar with annotation and amalgamation facilities please refer to appendix C

```
<metainfo> ::= (
                 ( <attribute_value_tag> | <hyperlink> )
                 '(' <probability> ')'
               )*
```

Definition Let the alphabet of wiki language $\mathcal{L}_W$ be a superset of F-logic alphabet $\Sigma_{\mathcal{F}}$. Let further q_{begin} be a regular expression that matches all query words beginnings, q_{formula} be a regular expression that matches possible F-logic formulas,[4] $q_{\text{delimiter}}$ a regular expression that matches delimiter words, $q_{\text{minimal probability}}$ a regular expression that matches a minimal probability word, $r_{\text{semantic template}}$ a regular expression that matches semantic templates, and q_{end} be a regular expression that matches all query word's endings. Let the following set of relations hold:

$$\begin{aligned} q_{\text{formula}} &\notin q_{\text{delimiter}} \\ q_{\text{minimal probability}} &\notin q_{\text{delimiter}} \\ r_{\text{semantic template}} &\notin q_{\text{end}} \end{aligned}$$

Then an *annotated query* is defined with the following regular expression:

$$r_{aquery} = q_{\text{begin}} q_{\text{formula}} q_{\text{delimiter}} q_{\text{minimal probability}} q_{\text{delimiter}} r_{\text{semantic template}} q_{\text{end}}$$

The semantics of an annotated query are as follows: for each result result_i obtained by issuing the query defined by q_{formula} against the knowledge base of the wiki system (the domain D) which is annotated with a probability that is greater or equal to p interpret the semantic template defined by $r_{\text{semantic template}}$ by exchanging any occurrence of a variable with the corresponding value from result_i. In a semantic wiki context this means that if on a wiki page a query occurs, than the formula defined by the query will be issued as a query against the knowledge base defined by the meta data of the wiki system. Each result will force the wiki language interpreter to write the wiki text of the semantic

[4] Again we leave the possibility open if this regular expression will possibly match words that aren't annotated F-logic formulas since frame logic is more expressive than regular expressions.

wiki template by exchanging all variables in it with corresponding values obtained from the result.

Definition Let $r_{\text{attribute-value tag}}$ be a regular expression that matches attribute-value tags, $r_{\text{hyperlink}}$ be a regular expression that matches hyperlinks, and $r_{\text{annotation}}$ then the following regular expression matches annotated meta information.

$$r_{\text{annotated meta information}} = ((r_{\text{attribute-value tag}}|r_{\text{hyperlink}})r_{\text{annotation}})*$$

We are now able to define the annotated semantic wiki language $\mathcal{L}_{SW}$ as follows:

Definition Let $\mathcal{L}_W$ be a wiki language, $\mathcal{L}_{\mathcal{F}}$ a F-logic language, let r_{aquery} be a regular expression that defines annotated queries, and $r_{\text{annotated meta information}}$ be a regular expression that matches annotated meta information. An *annotated semantic wiki language* $\mathcal{L}_{SW}$ is the pair $(\mathcal{L}_W, \mathcal{L}_{\mathcal{F}})$ bridged through r_{aquery} and $r_{\text{annotated meta information}}$. $\mathcal{L}_W$ is called the wiki component of language $\mathcal{L}_{SW}$, and $\mathcal{L}_{\mathcal{F}}$ is called the semantic component. r_{aquery} and $r_{\text{annotated meta information}}$ are the interface between $\mathcal{L}_W$ and $\mathcal{L}_{\mathcal{F}}$.

In order to demonstrate the approach we will take the following (imaginary) example of a wiki system.[5] Presume we have a wiki project entitled "Pepperland" with two wiki pages entitled "Music" and "Purpose of life". The two pages are the objects of the particular domain "Pepperland". Lets further presume that we have six project members collaborating on this wiki, namely "John", "Paul", "Ringo", "George", "Max" and "Glove". An intelligent agent "Jeremy Hilary Boob Ph.D (nowhere man)" tries to reason about the domain, but as it comes out, the domain is inconsistent. The following table shows the different viewpoints of project members:

"Ad hoc, ad loc and quid pro quo.
So little time — so much to know!"

Due to the disagreement on different issues a normal (semantic wiki) query would yield at least questionable results. For instance, if the disagreement statements are

[5] All images, names and motives are taken from the 1968 movie "Yellow Submarine" produced by United Artists (UA) and King Features Syndicate.

Table 7.1: Viewpoints of "Pepperland" project members

	Music	Purpose of life
John	`class : harmonious sounds`	`main purpose : love`
Paul	`class : harmonious sounds`	`main purpose : love`
Ringo	`class : harmonious sounds`	`main purpose : drums`
George	disagrees to (`class : evil noise`)	`main purpose : love`
Max	`class : evil noise`	disagrees to (`main purpose : love`)
Glove	`class : evil noise`	`main purpose : glove`

ignored in frame logic syntax the domain would be represented with a set of sentences similar to the following:[6]

$$
\begin{aligned}
?o : ?class \quad &\leftarrow \quad ?o\ [\ \text{class} \rightarrow ?class\ ]. \\
o_M \quad &[\quad \text{class} \twoheadrightarrow \{\ \text{evil noise},\ \text{harmonious sounds}\ \}\ ;\ \text{title} \rightarrow \text{Music}\ ]. \\
o_P \quad &[\quad \text{main purpose} \twoheadrightarrow \{\ \text{glove},\ \text{love},\ \text{drums}\}\ ;\ \text{title} \rightarrow \text{Purpose of life}\ ].
\end{aligned}
$$

Thus a query asking for the class to which the object entitled "Music" belongs:

$$?\text{–}\ \ o_M\ :\ ?class$$

would yield two valid answers, namely "evil noise" and "harmonious sounds". Likewise if querying for the value of the "main purpose" attribute of object o_P e.g.

$$?\text{–}\ \ o_P\ [\ \text{main purpose} \rightarrow ?purpose\ ]$$

the valid answers would be "glove", "love" and "drums". But, these answers do not reflect the actual state of the social system, since one answer is more meaningful to the social system, then the others.

Nowhere man thinks hard and comes up with a solution. The project members form a social network of trust, as shown on figure 7.1.

The figure reads as follows: Ringo trusts Paul and John, Paul trusts John, John trusts George, George trusts John, Max trusts Glove and Glove doesn't trust anyone. Using the previously described PageRank algorithm nowhere man was able to order the

[6]For sake of simplicity some technical details, e.g. predefined classes and attributes have been omitted.

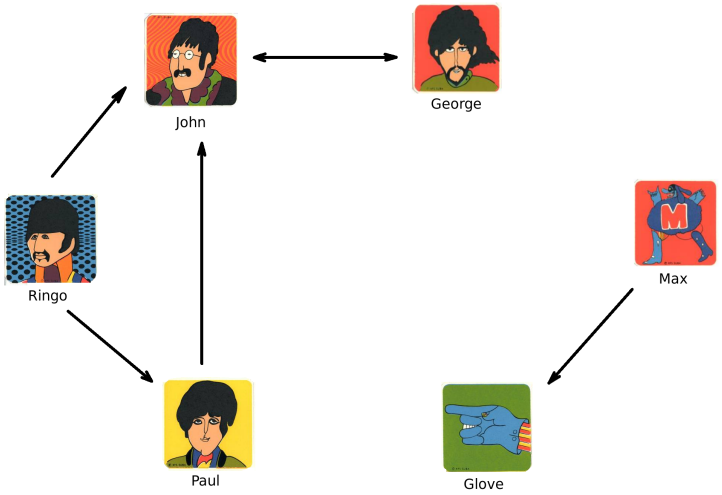

Figure 7.1: Social network of “Pepperland”

project members by their respective rank.

Table 7.2: Trust ranking of the “Pepperland” project members

Member	**Ranking**
John	0.303391
Glove	0.289855
George	0.267724
Paul	0.060667
Max	0.043478
Ringo	0.034884

Now, nowhere man uses these rankings to annotate the attribute-value tags given by the project members:

$$
\begin{aligned}
p(\text{class}) &= \text{Rank(John)} \\
&+ \text{Rank(Paul)} \\
&+ \text{Rank(Ringo)} \\
&- \text{Rank(George)} \\
&+ \text{Rank(Max)} \\
&+ \text{Rank(Glove)} \\
&= 0.303391 \\
&+ 0.060667 \\
&+ 0.034884 \\
&- 0.267724 \\
&+ 0.043478 \\
&+ 0.289855 \\
&= 0.464551
\end{aligned}
$$

As we can see the probability that object o_M is classified at all (e.g. not a generic object) is equal to the sum of project members rankings who agree to this statement (John, Paul, Ringo, Max and Glove) minus the sum of project members rankings who disagree (George). Note that if a member had tagged an object twice with the same attribute name, his ranking would be counted only once. Also note that if a member would have agreed and disagreed to an attribute name (e.g. disagreed to one attribute value but tagged another) his sum would be zero, since he would be at the agree and disagree side.

$$
\begin{aligned}
p(\text{evil noise}) &= \text{Rank(Max)} \\
&+ \text{Rank(Glove)} \\
&- \text{Rank(George)} \\
&= 0.065609
\end{aligned}
$$

From this probability calculation nowhere man is able to conclude that the formula o_M [class $\rightarrow$ evil noise] and likewise the formula o_M : evil noise holds with

probability $min(p(\text{class}), p(\text{evil noise}))$ which equals 0.065609. Likewise he calculates the probability of o_M : harmonious sounds

$$\begin{aligned} p(\text{harmonious sounds}) &= \text{Rank(John)} \\ &+ \text{Rank(Paul)} \\ &+ \text{Rank(Ringo)} \\ &= 0.398942 \end{aligned}$$

Since $min(p(\text{class}), p(\text{harmonious sounds}))$ equals 0.398942 he can now conclude that o_M : harmonious sounds holds more likely than o_M : evil noise with regard to the social network of project members. From these calculations nowhere man concludes that the final solutions to query $?\!-\ o_M : ?class$ are:

$$\begin{aligned} ?class &= \text{evil noise} \,\overline{\overline{\wedge}}\, 0.065609 \\ ?class &= \text{harmonious sounds} \,\overline{\overline{\wedge}}\, 0.398942 \end{aligned}$$

Nowhere man continues reasoning and calculate the probabilities for the other query:

$$\begin{aligned} p(\text{main purpose}) &= \text{Rank(John)} \\ &+ \text{Rank(Paul)} \\ &+ \text{Rank(Ringo)} \\ &+ \text{Rank(George)} \\ &- \text{Rank(Max)} \\ &+ \text{Rank(Glove)} \\ &= 0.913043 \end{aligned}$$

$$
\begin{aligned}
p(\text{love}) &= \text{Rank(John)} \\
&+ \text{Rank(Paul)} \\
&+ \text{Rank(George)} \\
&- \text{Rank(Max)} \\
&= 0.588304
\end{aligned}
$$

$$
\begin{aligned}
p(\text{glove}) &= \text{Rank(Glove)} \\
&= 0.289855
\end{aligned}
$$

$$
\begin{aligned}
p(\text{drums}) &= \text{Rank(Ringo)} \\
&= 0.034884
\end{aligned}
$$

From these calculations nowhere man concludes that $o_P\,[\,\text{main purpose} \rightarrow \text{love}\,]$ is most likely to hold with $p = 0.588304$. The final result of the query $?\!-\ o_P\,[\,\text{main purpose} \rightarrow ?purpose\,]$ is then:

$$
\begin{aligned}
?purpose &= love \mathbin{\overline{\overline{\wedge}}} 0.588304 \\
?purpose &= glove \mathbin{\overline{\overline{\wedge}}} 0.289855 \\
?purpose &= drums \mathbin{\overline{\overline{\wedge}}} 0.034884
\end{aligned}
$$

Now we can complicate things a bit to see the other parts of the approach in action. Assume now that John has created a link from the page entitled "Music" to the page entitled "Purpose of life", and named the link "has to do with". We would now have the following knowledge-base:

$$
\begin{array}{rcl}
?o : ?class & \leftarrow & ?o\ [\ \text{class} \rightarrow ?class\ ]. \\
o_M\ [& & \\
\text{class} & \twoheadrightarrow & \{\ \text{evil noise},\ \text{harmonious sounds}\ \}\ ; \\
\text{title} & \rightarrow & \text{Music}\ ; \\
\text{has to do with} & \rightarrow & o_P\ ]. \\
o_P\ [& & \\
\text{main purpose} & \twoheadrightarrow & \{\ \text{glove},\ \text{love}\ ,\text{drums}\}\ ; \\
\text{title} & \rightarrow & \text{Purpose of life}\ ].
\end{array}
$$

Now suppose that nowhere man wants to issue the following query:

$$?-\ ?o1\ :\ ?c\ [\ ?a \rightarrow ?o2\ ] \wedge ?o2[\ \text{main purpose} \rightarrow ?p\ ].$$

The solutions using "normal" frame logic are:

$$
\begin{array}{rcl}
s_1 : & & \\
?o1 & = & o_M \\
?c & = & \text{evil noise} \\
?a & = & \text{has to do with} \\
?o2 & = & o_P \\
?p & = & \text{glove}
\end{array}
$$

$$
\begin{array}{rcl}
s_2 : & & \\
?o1 & = & o_M \\
?c & = & \text{evil noise} \\
?a & = & \text{has to do with} \\
?o2 & = & o_P \\
?p & = & \text{love}
\end{array}
$$

s_3 :

$?o1 = o_M$

$?c =$ evil noise

$?a =$ has to do with

$?o2 = o_P$

$?p =$ drums

s_4 :

$?o1 = o_M$

$?c =$ harmonious sounds

$?a =$ has to do with

$?o2 = o_P$

$?p =$ glove

s_5 :

$?o1 = o_M$

$?c =$ harmonious sounds

$?a =$ has to do with

$?o2 = o_P$

$?p =$ love

s_6 :

$?o1 = o_M$

$?c =$ harmonious sounds

$?a =$ has to do with

$?o2 = o_P$

$?p =$ drums

To calculate the probabilities nowhere man uses the following procedure. The variables in the query are exchanged with the actual values for a given solution:

s_1 : o_M : evil noise [has to do with $\rightarrow$ o_P] $\wedge$ o_P[main purpose $\rightarrow$ glove].

s_2 : o_M : evil noise [has to do with $\rightarrow$ o_P] $\wedge$ o_P[main purpose $\rightarrow$ love].

s_3 : o_M : evil noise [has to do with $\rightarrow$ o_P] $\wedge$ o_P[main purpose $\rightarrow$ drums].

s_4 : o_M : harmonious sounds [has to do with $\rightarrow$ o_P] $\wedge$ o_P[main purpose $\rightarrow$ glove].

s_5 : o_M : harmonious sounds [has to do with $\rightarrow$ o_P] $\wedge$ o_P[main purpose $\rightarrow$ love].

s_6 : o_M : harmonious sounds [has to do with $\rightarrow$ o_P] $\wedge$ o_P[main purpose $\rightarrow$ drums].

Now according to rule 1 the conjunction becomes:

$p(s_1) = p(o_M$: evil noise [has to do with $\rightarrow$ o_P]) $\cdot$ $p(o_P$[main purpose $\rightarrow$ glove])

$p(s_2) = p(o_M$: evil noise [has to do with $\rightarrow$ o_P]) $\cdot$ $p(o_P$[main purpose $\rightarrow$ love])

$p(s_3) = p(o_M$: evil noise [has to do with $\rightarrow$ o_P]) $\cdot$ $p(o_P$[main purpose $\rightarrow$ drums])

$p(s_4) = p(o_M$: harmonious sounds [has to do with $\rightarrow$ o_P]) $\cdot$ $p(o_P$[main purpose $\rightarrow$ glove])

$p(s_5) = p(o_M$: harmonious sounds [has to do with $\rightarrow$ o_P]) $\cdot$ $p(o_P$[main purpose $\rightarrow$ love])

$p(s_6) = p(o_M$: harmonious sounds [has to do with $\rightarrow$ o_P]) $\cdot$ $p(o_P$[main purpose $\rightarrow$ drums])

The second parts of the equations were already calculated, and according to theorem 7.2.2 the first part of the equations becomes:

$$
\begin{aligned}
p(s_1) &= p(o_M : \text{evil noise}) \cdot \min(p(\text{has to do with}), p(o_P)) \cdot 0.289855 \\
p(s_2) &= p(o_M : \text{evil noise}) \cdot \min(p(\text{has to do with}), p(o_P)) \cdot 0.588304 \\
p(s_3) &= p(o_M : \text{evil noise}) \cdot \min(p(\text{has to do with}), p(o_P)) \cdot 0.034884 \\
p(s_4) &= p(o_M : \text{harmonious sounds}) \cdot \min(p(\text{has to do with}), p(o_P)) \cdot 0.289855 \\
p(s_5) &= p(o_M : \text{harmonious sounds}) \cdot \min(p(\text{has to do with}), p(o_P)) \cdot 0.588304 \\
p(s_6) &= p(o_M : \text{harmonious sounds}) \cdot \min(p(\text{has to do with}), p(o_P)) \cdot 0.034884
\end{aligned}
$$

We already know the probabilities of the is-a statement, and since

$$p(\text{has to do with}) = p(o_P) = Rank(\text{John}) = 0.303391$$

the equations become

$$
\begin{aligned}
p(s_1) &= 0.065609 \cdot 0.303391 \cdot 0.289855 \\
p(s_2) &= 0.065609 \cdot 0.303391 \cdot 0.588304 \\
p(s_3) &= 0.065609 \cdot 0.303391 \cdot 0.034884 \\
p(s_4) &= 0.398942 \cdot 0.303391 \cdot 0.289855 \\
p(s_5) &= 0.398942 \cdot 0.303391 \cdot 0.588304 \\
p(s_6) &= 0.398942 \cdot 0.303391 \cdot 0.034884
\end{aligned}
$$

and finally:

$$
\begin{aligned}
p(s_1) &= 0.005770 \\
p(s_2) &= 0.011710 \\
p(s_3) &= 0.000694 \\
p(s_4) &= 0.035083 \\
p(s_5) &= 0.071206 \\
p(s_6) &= 0.004222
\end{aligned}
$$

7.4 Amalgamation

To provide a mechanism for agents to query multiple annotated knowledge-bases we decided to use the principles of amalgamation. The model of knowledge base amalgamation is based on on-line querying of underlaying sources [51]. The intention of amalgamation is to show if a given solution holds in any of the underlaying sources.

Since the local annotations of different knowledge bases that are subject to amalgamation do not necessarily hold for the global knowledge base we need to introduce a mechanism to integrate the knowledge bases in a coherent way which will yield global annotations. Since the set of knowledge bases is a product of a set of respective social networks surrounding them, we decided to firstly integrate the social networks in order to provide the necessary foundation for global annotation.

Definition The integration of z social networks represented with the valued digraphs $(\mathcal{N}_1, \mathcal{A}_1, \mathcal{V}_1), ..., (\mathcal{N}_z, \mathcal{A}_z, \mathcal{V}_z)$ is given as the valued digraph $(\mathcal{N}_1 \cup ... \cup \mathcal{N}_z, \mathcal{A}_1 \cup ... \cup \mathcal{A}_z, \mathcal{V})$ where $\mathcal{V}$ is a function $\mathcal{V} : \mathcal{N}_1 \cup ... \cup \mathcal{N}_z \rightarrow \mathbb{R}$ that attaches values to nodes.

In particular $\mathcal{V}$ will be a social network analysis metric or in our case a variant of the eigenvector centrality. Now we can define the integration of knowledge bases as follows:

Definition Let $S_1, ..., S_z$ be sets of meta data statements as defined above representing particular knowledge bases in semantic wikis. The integration is given as $S_1 \cup ... \cup S_z$.

What remains is to provide the annotation that is at the same time the amalgamation scheme:

Definition Let $(\mathcal{N}_1 \cup ... \cup \mathcal{N}_z, \mathcal{A}_1 \cup ... \cup \mathcal{A}_z, \mathcal{V})$ be the integration of z social networks, let $S_1 \cup ... \cup S_z$ be the integration of their corresponding knowledge bases, and let $\rho : S_1 \cup ... \cup S_z \times \mathcal{V}$ be the relation that associates authors rankings to meta data statements, then the amalgamated annotation scheme $\overline{\overline{\wedge}}$ of the meta data statements is defined as follows:

$$s \,\overline{\overline{\wedge}}\, a_\Sigma, a_\Sigma = \sum_{(a,s)\in\rho} a$$

7.5 Amalgamated Annotated Semantic Wiki Language

To introduce amalgamation into annotated semantic wiki languages the definition of annotated query needs to be extended again.

Definition Let the alphabet of wiki language $\mathcal{L}_W$ be a superset of F-logic alphabet $\Sigma_{\mathcal{F}}$. Let further q_{begin} be a regular expression that matches all query words beginnings, q_{formula} be a regular expression that matches possible F-logic formulas,[7] $q_{\text{delimiter}}$ a regular expression that matches delimiter words, $q_{\text{minimal probability}}$ a regular expression that matches a minimal probability word, $q_{\text{amalgamation}}$ a regular expression that matches any list of knowledge base names to be amalgamated, $r_{\text{semantic template}}$ a regular expression that matches semantic templates, and q_{end} be a regular expression that matches all query word's endings. Let the following set of relations hold:

$$
\begin{aligned}
q_{\text{formula}} &\notin q_{\text{delimiter}} \\
q_{\text{minimal probability}} &\notin q_{\text{delimiter}} \\
q_{\text{amalgamation}} &\notin q_{\text{delimiter}} \\
r_{\text{semantic template}} &\notin q_{\text{end}}
\end{aligned}
$$

Then an *annotated query* is defined with the following regular expression:

[7] Again we leave the possibility open if this regular expression will possibly match words that aren't annotated F-logic formulas since frame logic is more expressive than regular expressions.

$$
\begin{aligned}
r_{aaquery} = \quad & q_{\text{begin}} \\
& q_{\text{formula}} \\
& q_{\text{delimiter}} \\
& q_{\text{minimal probability}} \\
& q_{\text{delimiter}} \\
& q_{\text{amalgamation}} \\
& q_{\text{delimiter}} \\
& r_{\text{semantic template}} \\
& q_{\text{end}}
\end{aligned}
$$

The semantics of an amalgamated annotated query are as follows: for each result result$_i$ obtained by issuing the query defined by q_{formula} against the set of knowledge bases defined in $q_{\text{amalgamation}}$ of different wiki systems (the domains $D_1, ..., D_z$) which is annotated with a probability that is greater or equal to p interpret the semantic template defined by $r_{\text{semantic template}}$ by exchanging any occurrence of a variable with the corresponding value from result$_i$. In a semantic wiki context this means that if on a wiki page a query occurs, than the formula defined by the query will be issued as a query against the knowledge base defined by the meta data of the wiki system. Each result will force the wiki language interpreter to write the wiki text of the semantic wiki template by exchanging all variables in it with corresponding values obtained from the result.

In the end we are now able to define amalgamated annotated semantic wiki languages as follows:

Definition Let $\mathcal{L}_W$ be a wiki language, $\mathcal{L}_\mathcal{F}$ a F-logic language, let $KB = \{KB_1, ..., KB_z\}$ be a set of knowledge bases, and let $r_{aaquery}$ be a regular expression that defines amalgamated annotated queries. An *amalgamated annotated semantic wiki language* $\mathcal{L}_{SW}$ is the pair $(\mathcal{L}_W, \mathcal{L}_\mathcal{F})$ bridged through $r_{aaquery}$ with regard to KB. $\mathcal{L}_W$ is called the wiki component of language $\mathcal{L}_{SW}$, and $\mathcal{L}_\mathcal{F}$ is called the semantic component. $r_{aaquery}$ is the interface between $\mathcal{L}_W$ and $\mathcal{L}_\mathcal{F}$ and applies to the set of knowledge bases KB.

To demonstrate the amalgamation approach proposed here lets again assume that our intelligent agent "Jeremy Hilary Boob Ph.D. (nowhere man)" tries to reason about the "Pepperland" domain, but this time he wants to draw conclusions from the domain "Yellow submarine" as well. The "Yellow submarine" domain is edited by "Ringo", "John", "Paul", "George" and "Young Fred" which form the social network shown on figure 7.2. Since the contents of this domain as well as the particular ranks of the members in it, won't be used further in the example they have been left out.

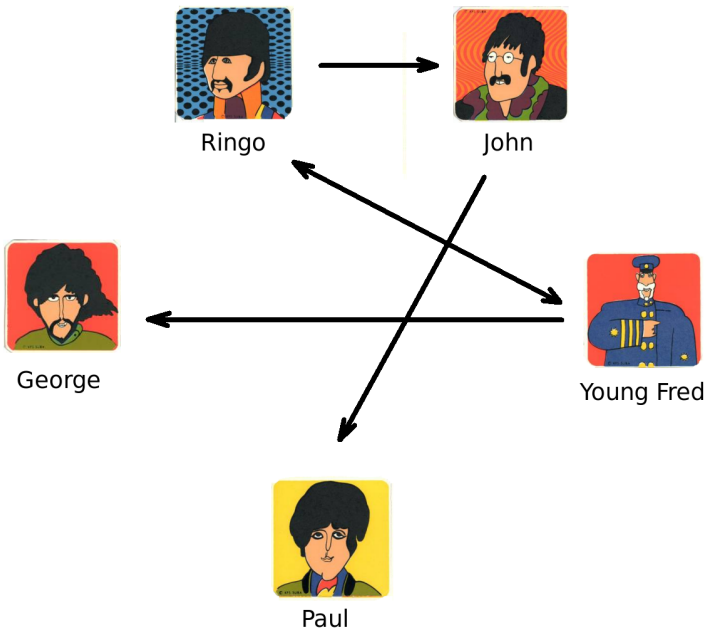

Figure 7.2: Social network of "Yellow submarine"

Since nowhere man wants to reason about both domains he needs to find a way to amalgamate these two domains.

"Where ground is soft, where often grows Arise, arouse, a rose a ... a rosy nose?"

Again he thinks hard, and comes up with the following solution. All he needs to do is to integrate the two social networks together, and recalculate the ranks of all members of this newly established social network in order to re-annotate the meta information in both domains.

Since the networks of "Pepperland" and "Yellow submarine" can be represented as the following sets of tuples:

$$
\begin{aligned}
\mathcal{G}_{\text{Pepperland}} = \{ & \\
& (\text{Ringo}, \text{John}), \\
& (\text{Ringo}, \text{Paul}), \\
& (\text{Paul}, \text{John}), \\
& (\text{John}, \text{George}), \\
& (\text{George}, \text{John}), \\
& (\text{Max}, \text{Glove}) \\
\} &
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{G}_{\text{Yellow submarine}} = \{ & \\
& (\text{Ringo}, \text{John}), \\
& (\text{Ringo}, \text{Young Fred}), \\
& (\text{John}, \text{Paul}), \\
& (\text{Young Fred}, \text{Ringo}), \\
& (\text{Young Fred}, \text{George}) \\
\} &
\end{aligned}
$$

All he needs is to find $\mathcal{G}_A = \mathcal{G}_{\text{Pepperland}} \cup \mathcal{G}_{\text{Yellow submarine}}$ and recalculate the ranks this new network. Thus

$$
\begin{aligned}
\mathcal{G}_A \quad = \quad & \{ \\
& (\text{Ringo}, \text{John}), \\
& (\text{Ringo}, \text{Paul}), \\
& (\text{Paul}, \text{John}), \\
& (\text{John}, \text{George}), \\
& (\text{George}, \text{John}), \\
& (\text{Max}, \text{Glove}) \\
& (\text{Ringo}, \text{Young Fred}), \\
& (\text{John}, \text{Paul}), \\
& (\text{Young Fred}, \text{Ringo}), \\
& (\text{Young Fred}, \text{George}) \\
\} &
\end{aligned}
$$

The newly established integrated social network is shown on figure 7.3.

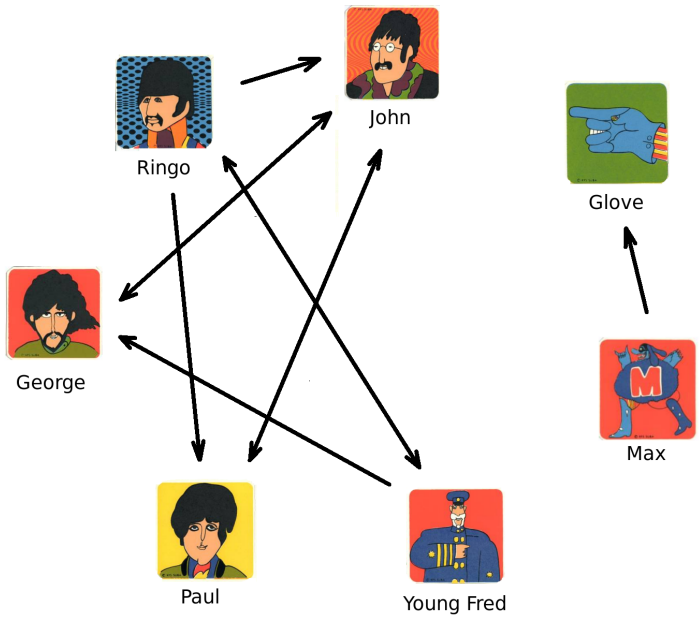

Figure 7.3: The integration of two social networks

Now nowhere man calculates the ranks of this new network and uses the previously described procedure to annotate the meta information (section 7.2) and reason about the amalgamated domain (section 7.2.1).

Chapter 8

The Niklas Language

Having the basic theory defined we tried to implement a semantic wiki language that will take into account the autopoietic social system surrounding it. The result is the **niKlas** language implemented into the ᴛAOPīs system. We need to make clear here that ᴛAOPīs is not only a semantic wiki system, but a systems for self-organizing communities and thus has other subsystems besides a semantic wiki system (like forums, blogging systems, mailing-lists, filtering systems, $\mathcal{F}$LORA-*2* and OWL export for possible agents etc.). This is why there is a small predefined vocabulary in **niKlas** that is shown on figure 8.1 was introduced.

Figure 8.1: Predefined class hierarchy in ᴛAOPīs

Basically there are three base classes defined (article, community, person) which have the following attributes:

article title, url, community, type, author

community id, description, founder

person name, surname, email, date_of_birth, date_of_registration, address, telephone

The subclasses differ only in their semantic context where they are interpreted. Nevertheless, niKlas doesn't need this vocabulary apart from TAOPIS , and can function well without it.

In the following we will show the syntax of niKlas through a set of examples reaching through all three components: wiki, semantic wiki, autopoiesis facilitating semantic wiki.

8.1 Wiki Component

8.1.1 Hyperlinks

We implemented four types of hyperlinks: normal, named, internal and inter-wiki hyperlinks. Normal hyperlinks use the following syntax:

```
[url]http://www.somewhere.org[/url]
```

Named hyperlinks are used to integrate a link inside some text in order to "hide' the URL and show a given name of the link. Such links have the following syntax:

```
[link=http://www.tiaktiv.hr>TiAktiv]
```

Internal hyperlinks are used to point to another page on the given project's or organization's wiki. They are also used to create new pages as discussed in chapter 2. The former part represents the title of the wiki page to which the link should point to, while the letter represents the name that will be shown. The syntax is as follows:

```
[link=Forum syntax>Formatting forum messages]
```

Internal inter-wiki hyperlinks point to wiki pages of other projects.[1] The first part is the name of the project/organization, the second part is the title of the wiki page and the third is the name to be displayed. The syntax is:

```
[link=TiAktiv:FrontPage - TiAktiv>Link to TiAktiv]
```

8.1.2 Images and Other Objects

niKlas allows users to include external images and YouTube movies in wiki pages. This functionality should be extended to allow the inclusion of other embedded objects. Images

[1]Note that TAOPIS is a project or organization hosting system that facilitates any project/organization with its own exclusive wiki.

can be included using the following syntax:

```
[img=http://www.foi.hr/logo.gif]
```

If one wants to edit the size of the image, additional parameters can be supplied as shown in the following example:

```
[img=http://www.foi.hr/logo.gif width=50% height=100]
```

Relative (%) or absolute sizes for both width and height can be used.

YouTube movies can be included in the following way:

```
[tube]o9698TqtY4A[/tube]
```

Where o9698TqtY4A is the code of the YouTube movie (one can get it from the URL on which the movie resides e. g. `http://www.youtube.com/watch?v=o9698TqtY4A`).

8.1.3 Headings

niKlas supports three levels of headings. Their syntax is simple and intuitive as the following three examples show:

Heading 2 syntax:

```
[h1]Heading 1[/h1]
```

Heading 2 syntax

```
[h2]Heading 2[/h2]
```

Heading 3 syntax

```
[h3]Heading 3[/h3]
```

8.1.4 Text Formatting

A number of different text formattings is supported by **niKlas** . A justified paragraph would be for example represented by the following syntax

```
[j]Paragraph content[/j]
```

Bold text can be used as follows:

```
[b]Bold text[/b]
```

Italic text corresponds to HTML:

```
[i]Italic text[/i]
```

As well as centered text does:

```
[center]Italic text[/center]
```

In order to provide tome programming code (where the usual HTML behavior is inconvenient - e.g. preformatted text neglected) one can use the code formatting syntax:

```
[code]Program code[/code]
```

It is sometimes convenient to quote some text as is the case in the following example:

```
[quote]Quoted text[/quote]
```

In order to mention the original author one can use:

```
[quote=Foo Bar]Named quoted text[/quote]
```

8.1.5 Lists and Tables

niKlas supports the creation of simple tables as well as two types of lists. The tables syntax was inspired by LaTeXas the following example shows:

```
[table]
row 1 column 1 && row 1 column 2 && row 1 column 3 ##
row 2 column 1 && row 2 column 2 && row 2 column 3 ##
row 3 column 1 && row 3 column 2 && row 3 column 3
[/table]
```

Unordered lists which can have up to three levels have the following syntax:

```
* one
** one one
** one two
*** one two one
*** one two two
*** one two three
* two
** two one
```

```
* three
```

Ordered lists are equivalent to unordered except that the level indicator sign is changed:

```
1 one
11 one one
11 one two
111 one two one
111 one two two
111 one two three
1 two
11 two one
1 three
```

8.1.6 Templates and Inclusion

The automated creation of a page outline and the inclusion of content from other pages is supported as well. An outline of forum posts or wiki pages can be created with:

```
[outline]
```

To include the content of another wiki page one just has to know the particular wiki page title as the following example shows:

```
[include WikiPageTitle]
```

To include a page from another project or organization the projects name has to precede the wiki page title:

```
[include ProjectName:WikiPageTitle]
```

Note that pages included from other projects or organizations are rendered locally, which means that links will point to the current project/organization. One should also have in mind that queries from included pages (local or external) are not rendered at all in the current implementation of **niKlas** .

8.1.7 References

References and citations are also supported. The syntax was as well inspired by LaTeX:

```
To use [cite references2009 references] and citations
use the following syntax:

[ref references2009] Luhmann, N: Soziale systeme, 1984.
```

8.2 Semantic Component

The **niKlas** syntax defines beside different text formatting commands, a query command:

```
[query=flora2_query. ]
{[header]header_formatting[/header]}
answer_formatting
[/query]
```

whereby *flora2_query* is a normal (restricted) $\mathcal{F}$LORA-*2* query with defined return variables, *header_formatting* is the optional header (possibly formatted using **niKlas** code),[2] and *answer_formatting* is a TAOPIS formatting that can contain variables used in the $\mathcal{F}$LORA-*2* query. The *answer_formatting* is repeated for any answer returned by the $\mathcal{F}$LORA-*2* reasoning engine by using the generated $\mathcal{F}$LORA-*2* ontology of the semantic wiki as a knowledge base.

As an illustrative example the following query would generate a list of users.

```
[query=?_:user[ name->?n ]. ]
[header][b]Users[/b][/header]
name: ?n
[/query]
```

The $\mathcal{F}$LORA-*2* query *?_:user[name->?n, surname->?s].* is issued against the dynamic knowledge base of the system. The obtained results are then replicated in the answer formatting; each answer prints out one answer formatting. On the other hand the header section will be printed only once. Thus the result of this query would be similar to:

```
[b]Users[/b]
```

[2] In the current version sub-queries are not allowed.

```
name: Markus
name: Mirko
name: Jurica
```

In the second phase, after the query generated a formatting, the rest of the **niKlas** code is translated to HTML but other target languages could be implemented. In this case, the HTML encoded answer would be:

```
<b>Users</b>
name: Markus<br />
name: Mirko<br />
name: Jurica<br />
```

In the following a few interesting use cases of dynamic queries shall be analyzed. We will show how **niKlas** can be used to dynamically generate new content that is usable in a wide range of community projects like current class hierarchies, dictionaries, FAQ's, tables of content, lists of editors, issue and bug tracking, as well as what links here links.

8.2.1 Class Hierarchies

In order to obtain a dynamically generated list of classes with corresponding subclasses one can issue the following query:

```
[query=?sub::?super. ]
?sub is a subclass of ?super
[/query]
```

The first line defines the *FLORA-2* query which is in particular a class expression with variables. In the second line the formating for the given list (which could have been a table or any other formatting) is defined. The third line closes the formatting.

Such a query will yield a list similar to:

```
apple is a subclass of fruit.
banana is a subclass of fruit.
fruit is a subclass of food.
```

Such a query would be useful if using a semantic wiki system as an ontology management tool, to provide a detailed overview of existing classes with corresponding

subclasses. There are variations to this query that would allow to obtain only strict subclasses, subclasses of a special class etc.

8.2.2 Dictionaries

Another interesting feature that can be provided using a dynamic query is a dictionary. Some wiki systems provide such a facility as a build-in function. Herein we show how such a functionality can be simulated in **niKlas** through a serious of queries similar to the following, provided that any wiki page is tagged with its corresponding first letter.[3]

```
[query=?_:wiki_page[
    title->?title,
    url->?address,
    letter->A ],
    sort(?title,asc). ]
[header][h1]A[/h1][/header]
[link=?address>?title]
[/query]
```

The query if provided on some wiki page would yield a result similar to the following whereby the corresponding titles would be links to the particular wiki pages dealing with them.

```
[h1]A[/h1]
[link=AnanasPage>Ananas]
[link=ApplePage>Apple]
...
```

To provide a full dictionary at the current version of **niKlas** the user has to make a query for any letter of a given alphabet. This is an unpleasant solution that implies that some future version of **niKlas** has to provide a facility for sub-queries and functionality similar to the SQL *group by* clause.

Such a query is usable in almost any wiki system. Especially in encyclopedia-like wikis such a query can be of great value.

[3]This tag is needed due to the fact that $\mathcal{F}$LORA-*2* syntax is a bit complex when it comes to string processing. A better solution using wildcards or regular expressions should be provided in some future version of **niKlas** .

8.2.3 Frequently Asked Questions

The popular FAQ section of some project or service can be simulated using a dynamic query. Provided that any wiki page that is an actual answer to a frequently asked question is tagged with attribute *question* and the corresponding question as its value, the query would look like the following.

```
[query=?_:wiki_page[
    question->?question,
    url->?address ],
    sort(?question,asc). ]
[link=?address>?question]
[/query]
```

The query would yield a sorted list of frequently asked questions with links to their answers similar to:

```
[link=ApplePie>What is apple pie?]
[link=PineApple>What is pineapple?]
...
```

Queries similar to this can be useful in almost any project or organization that interacts closely with their users/customers. Especially open source projects, customer relationship sites as well as others could have considerable benefits.

8.2.4 Tables of Content

In order to provide a book-like "linearization" of a wiki site on could provide a table of content using a series of queries similar to the following, provided that wiki pages are tagged with the chapter they belong to.

```
[query=?_:wiki_page[
    chapter->KM,
    title->?title,
    url->?address ],
    sort(?title,asc). ]
[header]Knowledge management[/header]
[link=?address>?title]
[/query]
```

Such a query would provide a list of sections of a given chapter with hyperlinks to the corresponding pages.

```
Knowledge management
[link=Explicit>Explicit knowledge]
[link=Tacit>Tacit knowledge]
...
```

In this case again niKlas current version shows a drawback for not supporting sub queries, since the user has to issue a query for any chapter.

Such a query (or serious of queries) is interesting in any documentation project, thesis, on-line book etc.

8.2.5 Who Edited this Page

One sometimes woulds like to know which users contributed to a given page. In niKlas this is achieved to a bit more complex query as follows.

```
[query=?_:wiki_page[
        title->'Page title',
        author->?_a ],
    ?_a:person[
        name->?name,
        surname->?surname,
        email->?email ]. ]
[header]Contributors:[/header]
?name ?surname
[url]mailto:?email[/url]

[/query]
```

By issuing this particular query we would obtain a list of users' names and surnames with links to their e-mail addresses.

```
Contributors:

Mirko Cubrilo
[url]mailto:mcubrilo@foi.hr[/url]
```

```
Markus Schatten
[url]mailto:mschatte@foi.hr[/url]

Jurica Seva
[url]mailto:jseva@foi.hr[/url]
```

Such queries are useful in any wiki site that wants to keep track of contributors to (for instance) facilitate the creation of a social network.

8.2.6 Issue Tracking

Lots of projects (especially information system based projects) have a particular need to keep track of issues that arise during the project (bugs, feature requests etc.). Provided that any wiki page that holds a bug description is tagged with *class:bug* as well as with *status:open* the following query would yield a list of open bugs.

```
[query=?_:bug[
    status->open,
    not( status->closed ),
    title->?title,
    url->?address ],
    sort(?title,asc). ]
[header]Open bugs[/header]
[link=?address>?title]
[/query]
```

The list would look similar to the following:

```
Open bugs
[link=BuildBug>Build crashes]
[link=ErrorOpen>Error on open]
...
```

In order to close an issue, a project member that solved it, just needs to tag the wiki page with *status:closed*, and the issue would be removed from the list. As indicated above, such a dynamic query can be useful on any project related site that deals with any kind of issues that one needs to keep track of.

8.2.7 What Links Here

A feature often included into conventional wiki systems is a list of pages that link to the current page. Provided that *'Page title'* is the title of the current page, this functionality can be simulated in **niKlas** as follows.

```
[query=?_:wiki_page[
      title->?title,
      url->?address,
      ?_->?_this_page ],
   ?_this_page:wiki_page[
      title->'Page title' ]. ].
[header]What links here?[/header]
[link=?address>?title]
[/query]
```

Thereby the result to this query would be a list of page titles with corresponding hyperlinks to the current page including links to them.

```
What links here?
[link=BananaPage>Banana]
[link=OrangePage>Orange]
...
```

This functionality is useful especially on wiki sites that have complex mutual hyperlinks (encyclopedia-like wikis, technical documentation etc.).

8.3 Autopoietic Component

8.3.1 Probability Annotation

To issue a query with minimal probability (example class hierarchy) one can use:

```
[query=?subclass :: ?superclass. ]
[probability > 0.1]
[header]
[b]SUBCLASS - SUPERCLASS[/b]
[/header]
[i]?subclass - ?superclass[/i]
```

```
[/query]
```

This query would retrieve all solutions which probability is higher that 0.1 with regard to the social network in which the query is executed. Thus, all other possible solutions will be discarded. The probability constraint can use any of the infix operators `>`, `<`, `<=` or `>=`. The querying engine takes care of all probability related matters. For implementation related issues please refer to appendix D.

8.3.2 Amalgamation

To issue a query on multiple amalgamated projects (example class hierarchy) use:

```
[query=?subclass :: ?superclass. ]
[amalgamate
        "Knowledge management"
        "Databases I"
        "Databases II"
]
[header]
[b]SUBCLASS - SUPERCLASS[/b]
[/header]
[i]?subclass - ?superclass[/i]
[/query]
```

The query would first amalgamate the knowledge bases of the projects "Knowledge management", "Databases I" and "Databases II" and then execute the query. In this way any number of projects can be amalgamated.

Amalgamation queries can be additionally constrained with probability, e.g.

```
[query=?subclass :: ?superclass. ]
[amalgamate
        "Knowledge management"
        "Databases I"
        "Databases II"
]
[probability > 0.4]
[header]
```

```
[b]SUBCLASS - SUPERCLASS[/b]
[/header]
[i]?subclass - ?superclass[/i]
[/query]
```

In this query all results with probability lower than 0.4 would be discarded. As in simple probability constrained queries other operators can be used.

8.4 A Short Comparison to other Semantic Wiki Engines

As a first one needs to state here that ᴛᴀOPīs is not primarily and only a semantic wiki system, as stated earlier. The system consists of different subsystems as indicated in [57]. The niKlas syntax can be used not only in the wiki subsystem, but in the forum, mailing list and blogging subsystems as well.

In order to provide a better understanding of what is new and different in ᴛᴀOPīs and likewise niKlas we will compare the system to the two maybe most elaborate semantic wiki engines at the time of writing this text: Semantic MediaWiki [89, 45, 46, 103] and IkeWiki [80, 81, 84]. The following table (8.1) summarizes the comparison.

Table 8.1: A comparison between Semantic MediaWiki, IkeWiki and ᴛᴀOPīs

Engine	**Semantic MediaWiki**	**Ike Wiki**	**ᴛᴀOPīs**
Markup syntax	MediaWiki	MediaWiki	Niklas
Access rights	Yes (via plug-in)	Yes	No
Tagging	Yes (inline)	Yes	Yes
Plugins	Yes	Yes	No
Rule support	Yes (via several plug-ins)	No (planned)	Yes (native)
Querying support	Yes	Yes	Yes
RDF/OWL export	Yes	Yes	Yes
Flora2 export	No	No	Yes
Filtering support	No	No	Yes
Dealing with uncertainty	No	No	Yes
Amalgamation	No	No	Yes

Maybe the most important difference between the ᴛᴀOPīs system and the two other outlined engines is the approach. ᴛᴀOPīs , due to its commitment to autopoiesis approaches its users as a social system, while Semantic MediaWiki and IkeWiki approach

individuals. This basic shift in viewpoints allows ꞮAOPĪs to define its users as a structurally coupling probabilistic system that is inconsistent in it self. This inconsistency is most likely to leave trails on the formalized knowledge accumulated on the semantic wiki, which is the main reason ꞮAOPĪs uses social network analysis to provide annotations to meta data.

As the table shows both Semantic MediaWiki and IkeWiki use MediaWiki syntax (known due to the famous Wikipedia) while ꞮAOPĪs developed its own **niKlas** syntax. The following listing shows some of the features of MediaWiki syntax:

```
{{SMW user TOC}}
The most important part of the [[Help:semantic search|Semantic ↘
→search]] features in [[Semantic MediaWiki]] is a simple format↘
→ for describing which pages should be displayed as the search ↘
→result. Queries select wiki pages based on the information ↘
→that has been specified for them using ''Categories', ''[[Help↘
→:Properties and types|Properties]]'', and maybe some other [[↘
→MediaWiki]] features such as a page's name-space. The ↘
→following paragraphs introduce the main query features in SMW.

== Categories and property values ==

In the [[Help:Semantic search|introductory example]], we gave ↘
→the single condition <nowiki>[[Located in::Germany]]</nowiki> ↘
→to describe which pages we were interested in. The markup text↘
→ is exactly what you would otherwise write to ''assert'' that ↘
→some page has this property and value. Putting it in a ↘
→semantic query makes SMW return all such pages. This is a ↘
→general scheme: ''The syntax for asking for pages that satisfy↘
→ some condition is exactly the syntax for explicitly asserting↘
→ that this condition holds.''
```

As one can see, MediaWiki syntax uses lots of special symbols to denote text formattings, links, categories etc. which are intermixed with HTML-like tags (< *nowiki* > for example), which makes is a bit hard to grasp for a new user. On the other hand **niKlas** uses self-explanatory HTML like syntax. The following example would yield similar effects

like the above in niKlas :

```
[h1]SMW user TOC[/h1]
The most important part of the [link=Help:semantic search>↘
→Semantic search] features in [link=Semantic MediaWiki>Semantic↘
→ MediaWiki] is a simple format for describing which pages ↘
→should be displayed as the search result. Queries select wiki ↘
→pages based on the information that has been specified for ↘
→them using [i]Categories[/i], [i][link=Help:Properties and ↘
→types>Properties][/i], and maybe some other [link=MediaWiki>↘
→MediaWiki] features such as a page's name-space. The following↘
→ paragraphs introduce the main query features in SMW.

[h2]Categories and property values[/h2]

In the [link=Help:Semantic search>introductory example], we ↘
→gave the single condition [+link=Located in>Germany+] to ↘
→describe which pages we were interested in. The markup text is↘
→ exactly what you would otherwise write to [i]assert[/i] that ↘
→some page has this property and value. Putting it in a ↘
→semantic query makes SMW return all such pages. This is a ↘
→general scheme: [i]The syntax for asking for pages that ↘
→satisfy some condition is exactly the syntax for explicitly ↘
→asserting that this condition holds.[/i]
```

Media wiki allows access rights (via an additional plug-in) while IkeWiki has native support for access rights. ᴛᴀOPīs lacks support for access rights due to its philosophy of TOP (transparent, open, public). No user has more right than another. Extra rights are gained through participation. Every user can edit any content, filter it, moderate the forum etc. The best editors will be awarded by higher rank, and their formalized meta data will be more important in reasoning. Additionally any user can use the moderation of another user, which allows the establishment of a top-list of best moderators for a given project. So, best moderators are awarded by more users using their moderation.

All three systems allow user-defined tags, but Semantic MediaWiki allows only

inline tags.[4] All three systems use these tags to generate meta data and formalize a given domain. ┬AOPÏS on the other hand seems to be the only one imposing a completely object-oriented framework [87], and does not describe presumed objects. Both Semantic MediaWiki and IkeWiki use presumed or predefined objects (like locations, years, metrics etc.) which in our opinion limits the expressiveness of the formal language since such objects cannot be redefined.

┬AOPÏS lack support for plug-ins or any user defined features. This is a major drawback of ┬AOPÏS that certainly has to be addressed in the future. Such plug-ins allowed Semantic MediaWiki and IkeWiki to extend their functionality in various ways. An interesting idea, which was outlined in this thesis, but not implemented in ┬AOPÏS , is the use of semantic web services to implement such additional functionality.

┬AOPÏS and Semantic MediaWiki have implemented rule support. While ┬AOPÏS has native support for rules, Semantic MediaWiki needs various plug-ins to allow its users to create rules [6]. For example, for a simple rule like "*if X is a brother of Y and is the father of Z then X is an uncle of Z*" in Semantic MediaWiki one has to write something like:

```
{{#arraymap:
  {{getValue|[[{{PAGENAME}}]]|brother of}}|,|Y|
  {{#arraymap:{{getValue|[[Y]]|father of}}|,|Z|
     [[uncle of::Z]]   }} }}
```

whereby *getValue* is a template with an ask query similar to:

```
{{#ask:[[{{{1}}}]] |?{{{2}}}=
  |mainlabel=- |format=list |link=none}}
```

The same could be achieved with ┬AOPÏS by tagging any page on a given wiki with:

```
rule : ?x[ uncle->?z ] :- ?_[ father->?x, brother->?z ].
```

All three wiki engines have inline querying facilities. While Semantic MediaWiki uses its own syntax, IkeWiki builds upon SPARQL[74] and ┬AOPÏS on $\mathcal{F}$LORA-*2* syntax. For example a query that yields all city names, population and area of cities located in Croatia would in Semantic MediaWiki look similar to:

```
{{#ask:
```

[4]Inline means inside wiki text. IkeWiki and ┬AOPÏS additionally allow tags separated from wiki content

```
        [[Category:City]]
        [[located in::Croatia]]
| ?population
| ?area#km2 = Size in km2
}}
```

The output format is built-in: a table. To change the output one has to use keywords or rather complex templates that have to be defined previously. In IkeWiki one has to use something similar to:

```
<?sparql
SELECT ?C ?P ?A
WHERE {
  ?C hasPopulation ?P .
  ?C occupiesArea ?A .
  ?C isLocatedIn "Croatia"
} ?>
```

IkeWiki's default output is a table as well, but it also allows user-defined formatting using predefined patterns.

```
<?sparql format=pattern pattern="YOUR PATTERN"
... SPARQL QUERY ...
?>
```

where *YOUR PATTERN* is arbitrary text with variable placeholders of the form {*V*} where *V* is the SPARQL variable to be substituted. In contrast the same query in niKlas would look similar to:

```
[query ?c:City[ location->Croatia, population->?p, area->?a ].]
?c has a population of ?p and occupies the area ?a.
[/query]
```

Thus the querying facilities of all three systems are of comparable quality. Also, all three systems allow the export of RDF/OWL ontology but only ⊤AOPīs allows export in $\mathcal{F}$LORA-*2* format.

Neither Semantic MediaWiki nor IkeWiki allow their users to organize their content through filtering. ⊤AOPīs allows filtering of individual pages, users, page versions etc.

Also neither Semantic MediaWiki nor IkeWiki allow dealing with uncertainties in any way nor amalgamating ontologies from different wikis, which is the main advantage of ᴛAOPīs . ᴛAOPīs allows conventional semantic wiki usage, but allows several advanced features. For example, it is quite obvious that on some topics users will disagree. On the other hand unexperienced users will likely make errors. Such disagreement and errors can yield inconsistencies outlined previously. These inconsistencies can be circumvented in a probabilistic environment. Users can pose queries that have probability constraints.

Further, there is quite often need to query various data sources. This perspective is often neglected by common semantic wiki systems which do not provide tools to manage such situations. ᴛAOPīs on the other hand allows the amalgamation of any number of wikis, and likewise annotating such amalgamated knowledge bases. In the current implementation wikis are amalgamated only through the amalgamation of the social networks of each wiki (which is in a way only a syntactic amalgamation). Users can take care of semantic amalgamation by imposing user defined equalities for example. The development of techniques for semantic amalgamation is subject to future research.

Chapter 9

Application Examples

In the following few sections we shall provide three possible application areas of the niKlas language as well as autopoiesis facilitating semantic wiki systems, but application is possible in any situation where a collaborative management of knowledge is appropriate.

9.1 Autopoietic System for Personal Computer Security

The idea of creating an open system for security seems paradoxical but possible. Popular open source software, due to the usage of lots of users, became very secure since it was able to leverage the knowledge of the social system. The same idea applies to personal computer (PC) security.

If we imagine a common semantic wiki system where users can add formalized knowledge about known security issues on a particular platform certain intelligent agents could be developed. Such agents need to be able to analyze the semantic content on the wiki system with regard to the particular PC configuration, and fix common issues using the semantic content.

On the other hand, malicious users could try to compromise the semantic wiki system, due to its openness, in order to do harm or gain access to users PC-s. To prevent such possibilities the use of potentially malicious formalized knowledge has to be minimized.

To do so the social network has to be formalized with trust relations between users. Such trust relations will help in constructing a dynamic hierarchy of most trusted contributors with their respective trust-ranks. The adequate trust level needs to be deter-

mined empirically, so that agents can be calibrated to issue queries that will return only trustworthy results and not compromise the users system.

As a simple prototype example, assume that we have a semantic wiki system entitled "SecureAIS" that deals with security issues for a piece of software entitled foo. Further assume that 3 users participate in this wiki (each using a different graphical user interface[1] on the following figures):

- Foo Developer - a trusted advanced user that provides patches for foo.
- Foo User - a normal user that needs to take care of his installation of foo.
- Foo Attacker - a malicious user trying to take advantage of Foo User.

Foo User thrusts Foo Developer, and this is the only thrust relation in the network. Thus the respective thrust ranks of SecureAIS members are as shown on figure 9.1 (from a Foo User perspective).

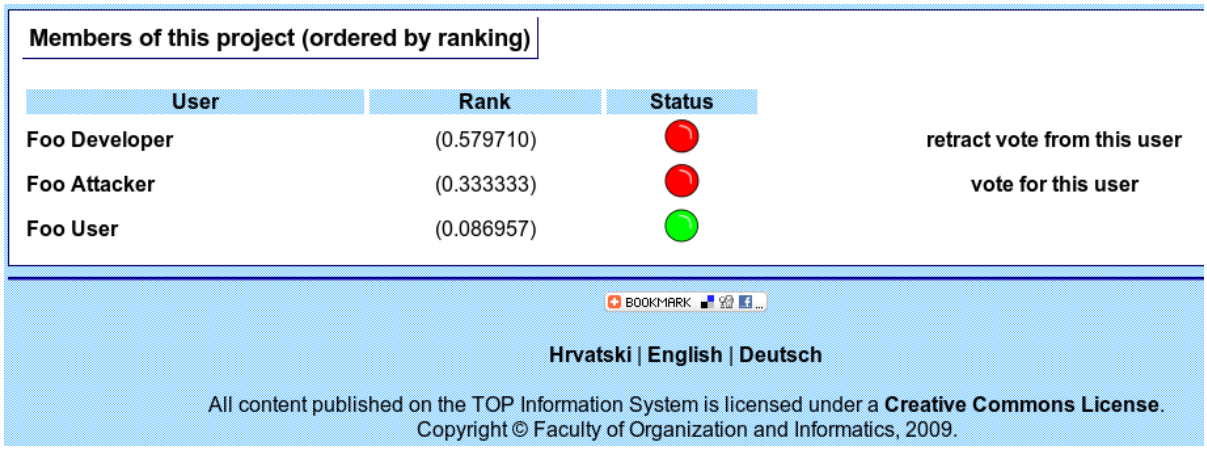

Figure 9.1: Ranks of the SecureAIS semantic wiki project members

Assume further that the semantic wiki consists of objects (pages) of the class patch that have attributes `software` (the name of the software product the patch applies to), `version` (the version it applies to), `depends` (an optional attribute that indicates that a patch depends on some other patch) and `file` (the url of the file that contains the patch). In essence wiki pages will be tagged with the first three attributes, and will contain a link to the patch file. Figure 9.2 shows the class diagram of these objects.

Foo Developer provided the following two rules that ease querying for patches by tagging the front page with attribute rule and the appropriate $\mathcal{F}$LORA-*2* code:[2]

[1] ┬AOPīs allows users to use different GUI-s when logged in.
[2] The code is checked by ┬AOPīs and invalid rules are discarded from the knowledge base.

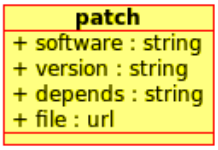

Figure 9.2: UML diagram of SecureAIS

```
?x[ dependency->?x ] :-
  ?x:patch.

?x[ dependency->?y ] :-
  ?x[ depends->?title ],
  ?y[ title->?title ].
```

The first rule makes sure that every patch depends on it self assuring thereby that the latest patch is included in a possible answer set. The second rule connects patches through their dependency attributes.

Foo Developer now goes on and provides his first patch (figure 9.3).

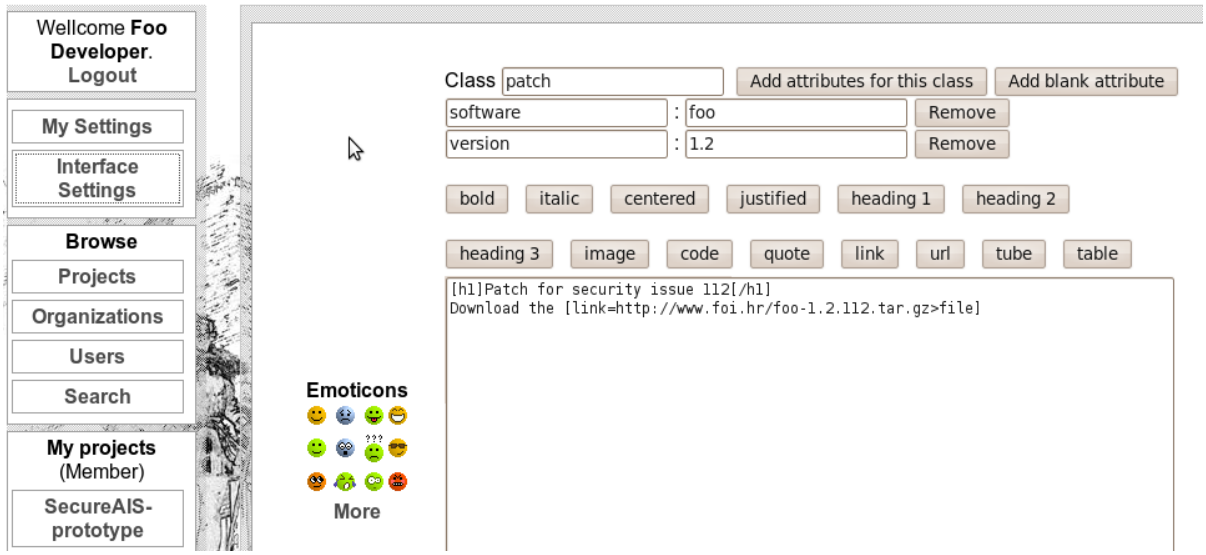

Figure 9.3: Adding the first patch to SecureAIS

ᴛAOPĪS provides him with the means to add tags at page creation. The wiki page for this patch then would look similar to the following figure (9.4).

By adding the second patch the system turns on suggestions (figure 9.5) for possible classes. Once a class has been chosen/entered, the user can add all attributes for this class, by clicking the appropriate button. By typing in the values of attributes, a suggestion mechanism shows already given values, preventing thereby possible syntax

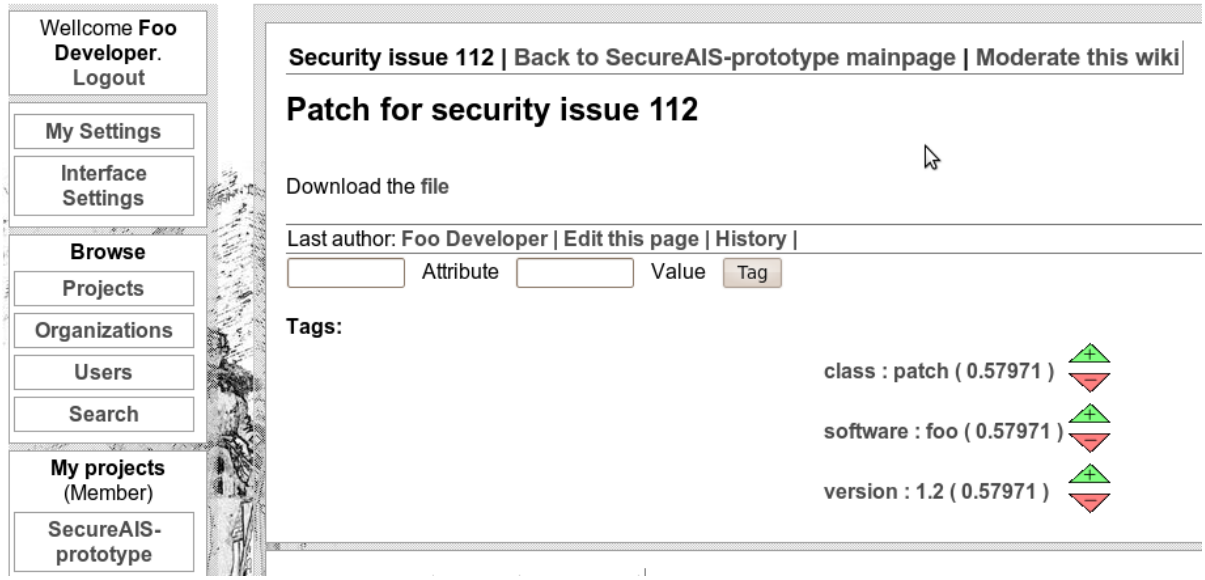

Figure 9.4: Wiki page of the first patch on SecureAIS

errors. This suggestion mechanism applies to all user entered meta data forms.

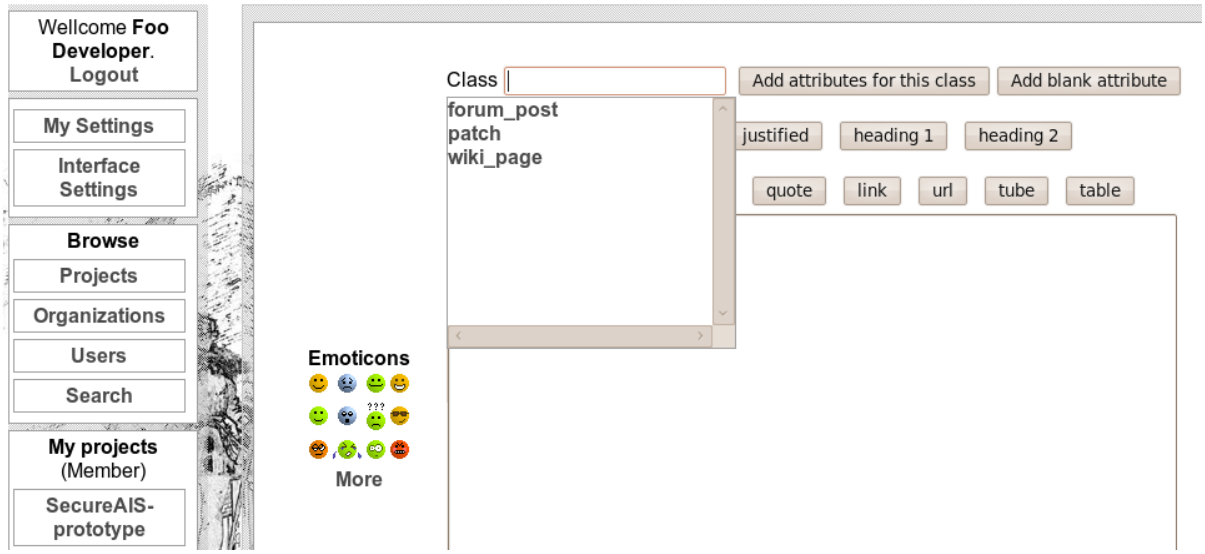

Figure 9.5: Adding the second patch to SecureAIS

Assume that Foo Developer added a total of three pages tagged as follows:

- `Security issue 112`
  - `class : patch`
  - `software : foo`
  - `version : 1.2`
- `Security issue 120`
  - `class : patch`

- software : foo
- version : 1.2
- depends : Security issue 112

• Security issue 123

- class : patch
- software : foo
- version : 1.2
- depends : Security issue 120

Additionally every page has a link to the patch file entitled `file`. Thus the following three objects are part of the SecureAIS knowledge base:

```
oid_1:patch[
  title->'Security issue 112',
  software->foo,
  version->'1.2',
  file->'http://www.foi.hr/foo-1.2.112.tar.gz'
].

oid_2:patch[
  title->'Security issue 120',
  software->foo,
  version->'1.2',
  depends->'Security issue 112',
  file->'http://www.foi.hr/foo-1.2.120.tar.gz'
].

oid_3:patch[
  title->'Security issue 123',
  software->foo,
  version->'1.2'
  depends->'Security issue 120',
  file->'http://www.foi.hr/foo-1.2.123.tar.gz'
].
```

Now Foo Attacker adds also a page to the wiki, tagging it identically to other patches on the system (figure 9.6). The only difference is that the patch file is a virus.

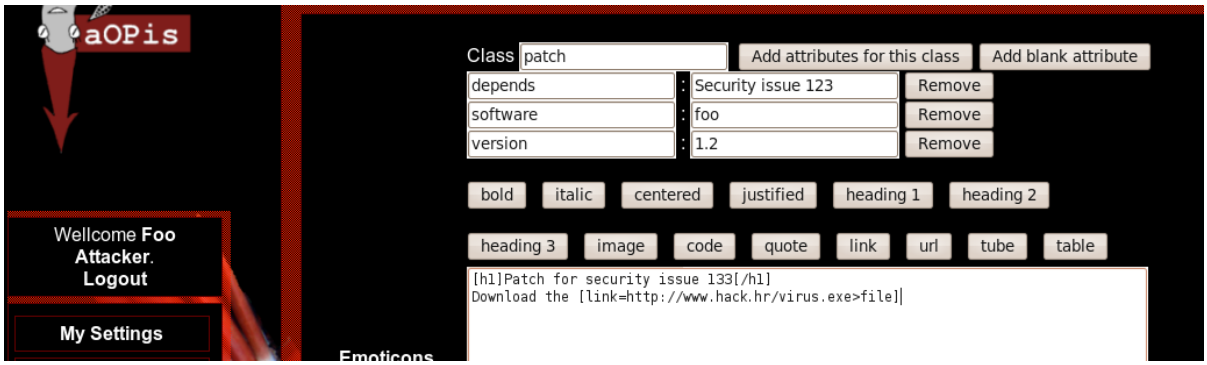

Figure 9.6: Attacker adding virus to SecureAIS

As shown on the following figure 9.7, the page looks almost exactly like the normal patch pages.[3]

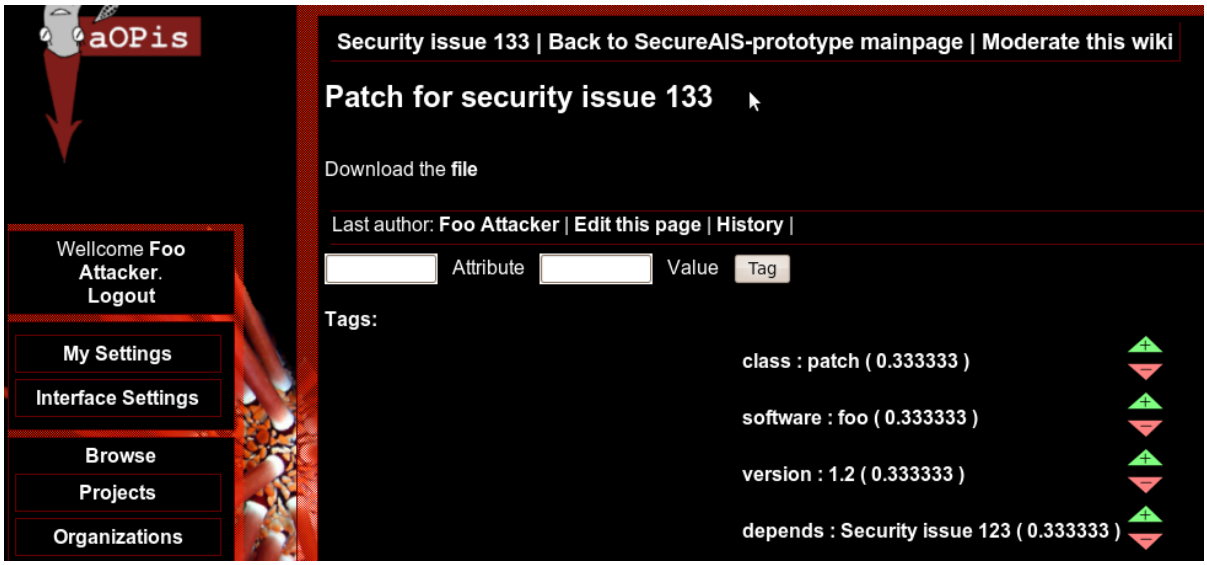

Figure 9.7: Malicious patch wiki page

Thus an additional object is added to the wiki knowledge base:

```
oid_4:patch[
  title->'Security issue 133',
  software->foo,
  version->'1.2',
  depends->'Security issue 123',
  file->'http://www.hack.hr/virus.exe'
].
```

[3]Except that Foo Attacker is using a different GUI.

If Foo User now wants to update his installation of foo be issuing a query in **niKlas** similar to:

```
[query
  ?_:patch[
    software->foo,
    dependency->?_d
  ],
  ?_d:patch[
    file->?patch
  ]. ]
[header]Patches to download[/header]
?patch
[/query]
```

He would acquire all patches including the malicious one. But, since there is a social network behind the semantic wiki system all objects are annotated, and thus Foo User can use an annotated query to receive only trustful patches. The minimal probability or thrust level needs to be determined empirically, as stated earlier, but in this case a probability greater than 0.2 will do the trick. Thus the following query will filter out the malicious patch, as shown on figure 9.8.

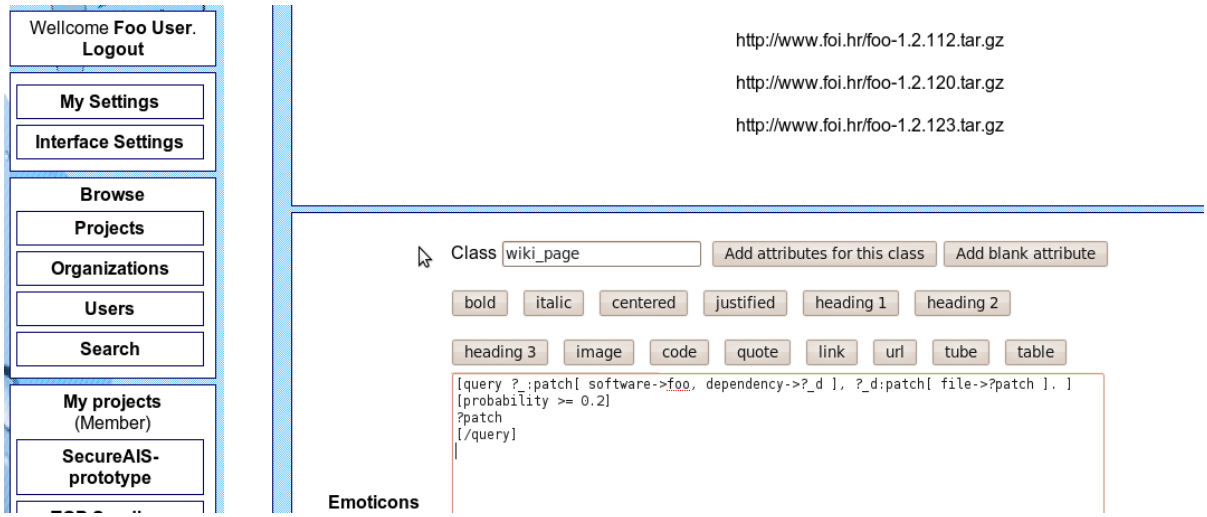

Figure 9.8: Query with malicious patches filtered out

Off course, the example is a bit simplistic, but SecureAIS can be enriched to support different software, versioning, anti-virus protection, SPAM filtering etc., and various tasks can be automated using intelligent agents, web services etc. but this isn't

the subject of this thesis.

Similar systems are already in wide use, and known as package managers mostly used in different open source Linux operating systems like Debian or Ubuntu. They allow users to get security fixes and software updates whenever they please. The interesting fact about these systems is that most software updates are written by enthusiasts and voluntaries. Users implicitly trust such software without any formal organization standing behind them. Users trust a social network of developers and believe that they wouldn't compromise their systems.

9.2 Autopoietic Scientific Publishing System

Scientific publishing is another possible application area. The process of reviewing submissions to respective conferences and journals (which mostly are double-blind review) could be established in an autopoietic environment using semantic wiki systems. On the other hand publishing technology has advanced from ordinary typography and it is ironic that scientific institutions, which should be the primers in using advanced technologies, still use traditional (often black and white) paper publishing. Also web interfaces to journals act mainly as a digital archive of non-digital papers. Technology allows us to use multimedia systems, software applications, social and semantic web facilities that could tremendously improve publishing quality and allow for more scientific interaction.

A semantic wiki system could be used to create a scientific publishing system. Any article would be published immediately after submission and would then undergo a continuous review process. Any user could review any encountered article, suggest improvements, add keywords, point out related research etc. Additionally authors could provide contemporary multimedia content like video, animations, interactive application examples or even recorded presentations.

The use of a formal semantic system could allow for better search and retrieval of scientific content. For example references could be formalized and thus citation indexes could be easier automatically computed. User provided meta data could be used by intelligent agents to identify breakthrough research.

On the other hand to ensure scientific seriousness, meta data and reviews would be rated depending on a social network analysis subsystem. Social network analysis could also be used to identify most prominent scientists.

As an example a prototype entitled "Journal of Publish or Perish" will be

shown. Figure 9.9 shows the front page of the scientific journal wiki.

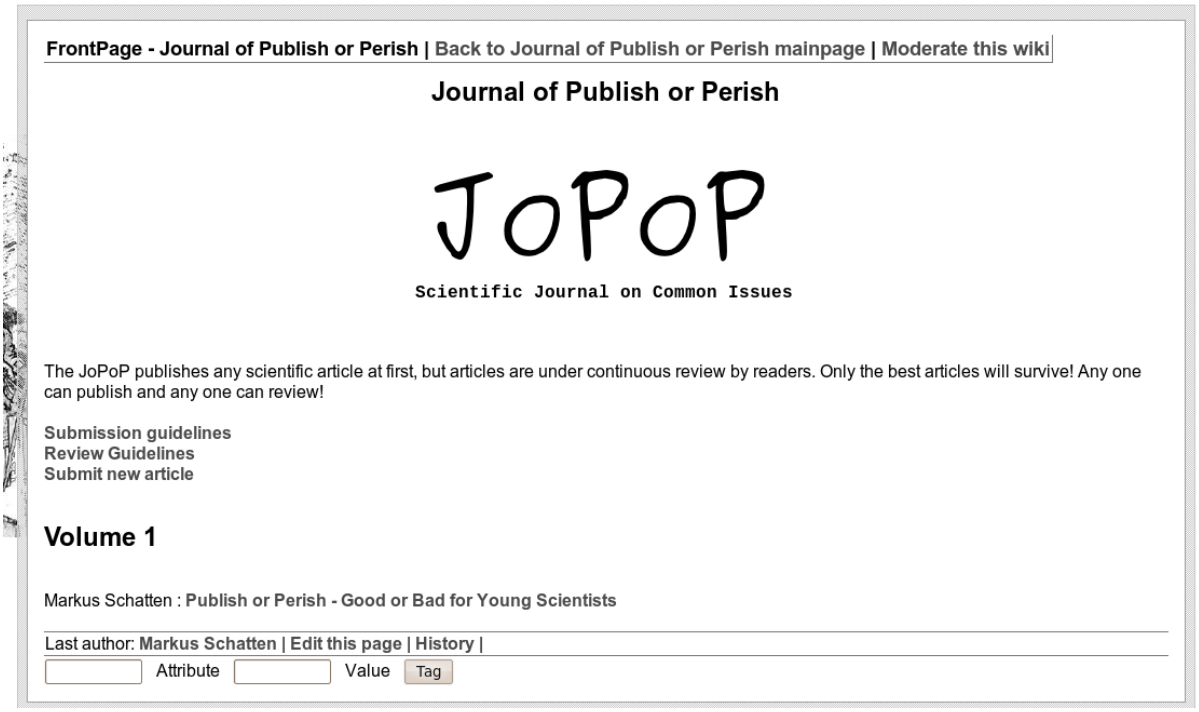

Figure 9.9: Frontpage of JoPoP

The following listing shows the niKlas code used to generate this frontpage. Note the query used to generate the table of content for the first volume.

```
[center][h1]Journal of Publish or Perish[/h1]
[img=http://arka.foi.hr/~mschatten/slike_taopis/logo_jopop.png]
[/center]

The JoPoP publishes any scientific article at first, but ↘
→articles
are under continuous review by readers. Only the best articles
will survive! Any one can publish and any one can review!

[link=Guidelines>Submission guidelines]
[link=Review Guidelines>Review Guidelines]
[link=Enter title>Submit new article]

[h1]Volume 1[/h1]
[query
  ?_:paper[
```

```
    author->?_aut ,
    title ->?t ,
    url->?url ,
    volume->1
  ],
  ?_aut:person[
    name->?name,
    surname->?surname
  ].
]
?name ?surname : [link=?url>?t]
[/query]
```

By clicking on the *submit new article* link a potential author can format his manuscript as any wiki page, as shown on the following figure 9.10. The author can add various keywords, but other meta information as well.

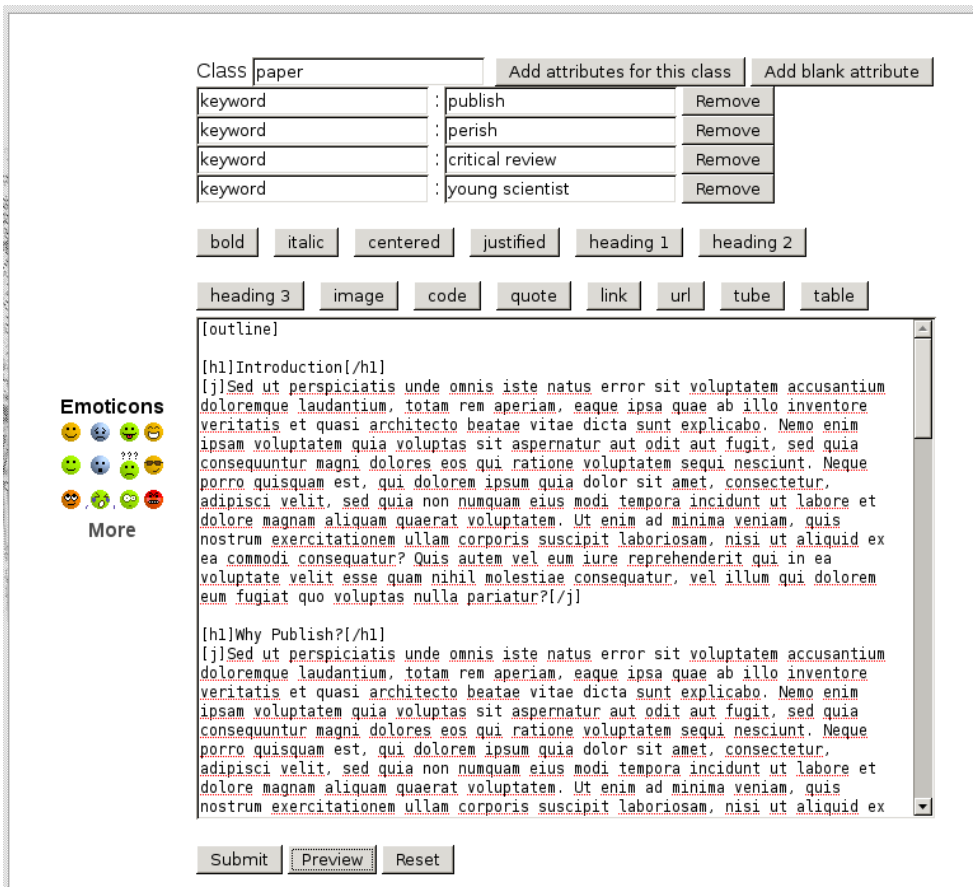

Figure 9.10: Submitting a new manuscript

This example manuscript would look similar to the following figure 9.11.

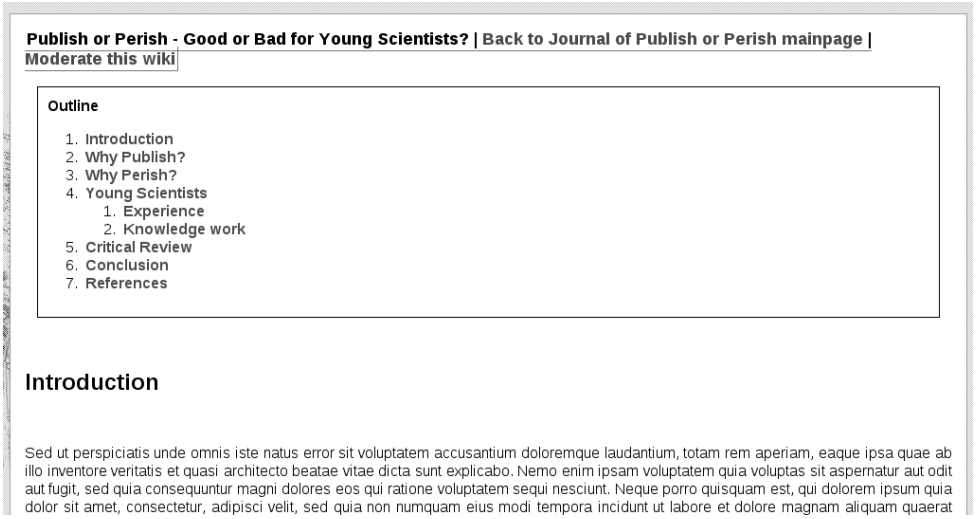

Figure 9.11: A manuscript on JoPoP

Due to the possibilities of **niKlas** (or any other wiki language) the author can add multimedia content at will. Figure 9.12 shows an example where a YouTube movie was included into to the manuscript as a figure by using the following **niKlas** code:

```
[center][tube]o9698TqtY4A[/tube]
[b]Figure 1.[/b] What happens to young scientists when they ↘
→cannot publish[/center]
```

It would be convenient to implement some automatic mechanism for figure and section numbering. Due to the page inclusion mechanism of **niKlas** (subsection 8.1.6) it would be possible to externalize multimedia content and provide it with additional meta information.

Figure 9.12: Multimedia on JoPoP manuscript

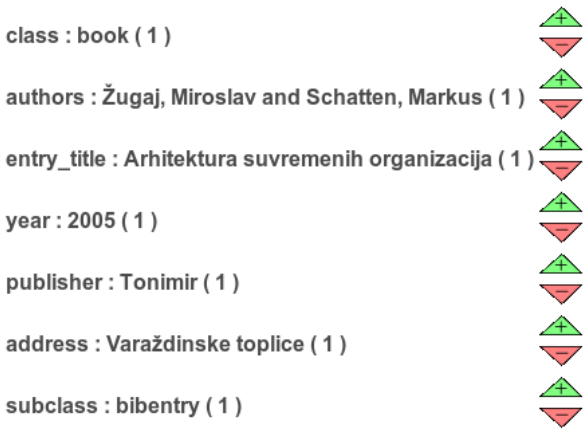

Figure 9.13: Tags on a sample bibliographic entry

The semantic wiki can also be used to automatically generate references similar to LaTeXand BiBtex. For example if references are objects (wiki pages) of their own, tagged similar to BiBtex entries as shown on figure 9.13, and likewise if references in some paper are given with the following niKlas code:

```
[link=ZugajSchatten2005>citation]
```

where `ZugajSchatten2005` is the title of the reference, than the following query would yield the list of references for a given paper with hyperlinks to the entry.

```
[query
  ?_[
    title->'Publish or Perish - Good or Bad for Young ↘
    →Scientists',
    citation->?_b
  ],
  ?_b:bibentry[
    authors->?a,
    year->?y,
    entry_title->?t,
    publisher->?p,
    address->?d
  ],
  sort(?a,asc). ]
[b]?a[/b] (?y) [i]?t[/i], ?p, ?d.
[/query]
```

On our imaginary paper the result of this query would look similar to the following figure 9.14

Figure 9.14: References generated by a query

If the wiki is used only as a bibliographic database, then the following query could be interesting to LaTeX users:

```
[query
  ?_:book[
    title->?r,
    authors->?a,
    entry_title->?t,
    year->?y,
    publisher->?p,
    address->?d
  ],
  sort(?a,asc).
]
@book{ ?r,
  title = "?t",
  author = "?a",
  publisher = "?p",
  address = "?d",
  year = "?y"
}
[/query]
```

The result of this query is shown in the following listing, and is a list of all books on a system in BiBtex format. Similar queries could be constructed to list all other types of bibliographic entries.

```
@book{ ZugajSchatten2005,
  title = "Arhitektura suvremenih organizacija",
```

```
  author = "\v{Z}ugaj, Miroslav and Schatten, Markus",
  publisher = "Tonimir",
  address = "Vara\v{z}dinske toplice",
  year = "2005"
}
```

The amalgamation facility could be the means of using one bibliographic semantic wiki through various semantic wiki journals. For example if we assume that the bibliographic wiki is entitled "BibWiki" then the previous query for bibliography creation would be slightly modified to yield the proper results:

```
[query
  ?_[
    title->'Publish or Perish - Good or Bad for Young ↘
    →Scientists',
    citation->?_b
  ],
  ?_b:bibentry[
    authors->?a,
    year->?y,
    entry_title->?t,
    publisher->?p,
    address->?d
  ],
  sort(?a,asc). ]
[amalgamate
  "Journal of Publish or Perish"
  "BibWiki"
]s
[b]?a[/b] (?y) [i]?t[/i], ?p, ?d.
[/query]
```

Any person could add reviews on any paper. If we establish a procedure that any review has to have a link to the paper it reviews entitled `on`, and if all reviews have to be tagged as shown on figure 9.15.

Then a query like the following could be used on any paper to list the reviews

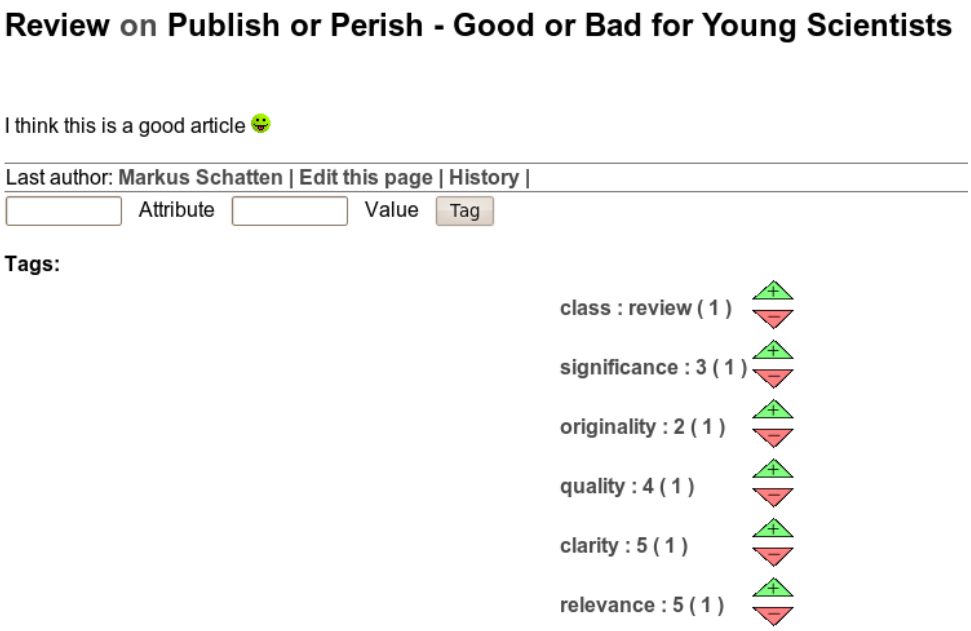

Figure 9.15: A tagged review

of the paper.

```
[h1]Reviews[/h1]
[center]
[table]
[b]Reviewer[/b]&&significance&&originality&&quality&&clarity&&↘
→relevance##
[query
  ?_:review[
    on->?_pap,
    significance->?s,
    originality->?o,
    quality->?q,
    clarity->?c,
    relevance->?r,
    author->?_aut,
    url->?url
  ],
  ?_pap:paper[
    title->'Publish or Perish - Good or Bad for Young ↘
    →Scientists'
  ],
```

```
  ?_aut:person[
    name->?name,
    surname->?lname
  ].
]
[link=?url>?name ?lname]&&?s&&?o&&?q&&?c&&?r##
[/query]
[/table]
[/center]
```

The result of such a query would be similar to the one shown on figure 9.16.

Reviews

Reviewer	significance	originality	quality	clarity	relevance
Markus Schatten	3	2	4	5	5

Figure 9.16: List of query generated reviews

By using more complex queries with aggregate functions average grades could be calculated. Such queries can become quite cumbersome due to the fact that all tags are stored as strings, and need to be converted to numbers. The following is an example of calculating the average quality grade:

```
[query
  ?avg = average{
    ?_q |
      ?_[ quality->?_s ],
      name(?_s,?_x)@_prolog,
      number_codes(?_q,?_x)@_prolog
  }.
]
?avg
[/query]
```

On the other hand the annotation mechanism could be used as well in order to find only those reviews or articles that are relevant to a certain degree. As an example

the following query would yield only reviews with trust-level higher that 0.5.

```
[h1]Reviews[/h1]
[center]
[table]
[b]Reviewer[/b]&&significance&&originality&&quality&&clarity&&relevance##
[query
  ?_:review[
    on->?_pap,
    significance->?s,
    originality->?o,
    quality->?q,
    clarity->?c,
    relevance->?r,
    author->?_aut,
    url->?url
  ],
  ?_pap:paper[
    title->'Publish or Perish - Good or Bad for Young Scientists'
  ],
  ?_aut:person[
    name->?name,
    surname->?lname
  ].
]
[probability > 0.5]
[link=?url>?name ?lname]&&?s&&?o&&?q&&?c&&?r##
[/query]
[/table]
[/center]
```

Such a query would filter out all lowly trusted reviewers, and could be used as a filter for published or perished articles.

9.3 Autopoietic Knowledge Management System

Modern organizations, as outlined in chapter 6, have new needs and need to leverage their knowledge faster then ever before. The construction of sophisticated knowledge bases, decision support systems as well as other intelligent systems often takes time (and money) but doesn't yield results as fast as needed.

Using a semantic wiki system a self-organizing corporate knowledge base could be constructed by letting employees interact with the system and formalize their knowledge about business. Common Enterprise 2.0 systems already use the lessons learned from Web 2.0, but by introducing semantic technologies such systems could be improved.

Through the use of social networks natural leaders could be identified, and due to the fast information flows new opportunities could be dealt with sooner. Simple querying mechanisms could be developed to facilitate managers with decision support. By connecting such a system to existing databases and information systems through web services as outlined before an integral knowledge management solution could be established that would reflect the current state of the organization and its environment.

Suppose, for example, that some organization "X" consists of five organizational units:

- Marketing
- Sales
- Accounting
- Production
- Human resource

Each department has its own knowledge base in form of a semantic wiki system. Suppose further that all employees can tag their selves and other employees with their skills.[4] For example, a sales employee could be tagged as:

- `skill : presentation`
- `skill : communication`
- `skill : management`

[4] ┬AOPIs allows also the tagging of users in addition to tagging articles.

A manager wants to get an overview of the employees skills. A query similar to the following would do the job for him:

```
[query
  ?dept:organization,
  ?skills = collectset{
    ?_y |
      ?_:person[
        skill->?_y,
        member_of->?dept ]
  },
  sort(?dept,asc). ]
[amalgamate
  "Marketing"
  "Sales"
  "Accounting"
  "Production"
  "Human resource"
]
?dept --> ?skills
[/query]
```

The query would yield a list similar to the following:

```
Marketing --> [communication, presentation, design]
Sales --> [communication, presentation, negotiation]
Accounting --> [finance, excel]
Production --> [database, Linux, presentation, programming]
Human resource --> [communication, databases, management]
```

Or, for example, if a HR manager would like to know if any department hasn't got any communication skills, a query similar to the following could be issued:

```
[query
  ?dept:organization,
  ?_y = collectset{
    ?_y |
      ?_:person[
```

```
        skill->?_y,
        member_of->?dept
      ]
  },
  not(
    member(communication,?_y)@_prolog(basics)
  ),
  sort(?dept,asc).
]
[amalgamate
  "Marketing"
  "Sales"
  "Accounting"
  "Production"
  "Human resource"
]
?dept
[/query]
```

The query would yield a list of departments who are in desperate need for a communication skills seminar.

Suppose further, for example, that employees are tagged with their current projects they work on. The following query could provide a manager with a list of employees that work on more than 4 projects, and need to be sent on vacation:

```
[query
  ?_e:person[
    name->?name,
    surname->?surname
  ],
  ?projects = collectset{
    ?_p |
      ?_e[
        project->?_p
      ]
```

```
  },
  ?_count = count{
    ?_x |
      member(?_x,?projects)@_prolog(basics)
  },
  ?_count > 4,
  sort(?surname,asc).
]
[amalgamate
  "Marketing"
  "Sales"
  "Accounting"
  "Production"
  "Human resource"
]
?name ?surname : ?projects
[/query]
```

The result is, as expected, shown on figure 9.17.

Figure 9.17: List of employees working on more than four projects

We could also easily build a simple agent querying projects on one ore more TAOPIS instances. The following listing presents a simple Python script that downloads a given projects knowledge base.

```
# -*- coding: utf-8 -*-
import urllib
import re
import sys

if len( sys.argv ) > 1:
```

```
  url = sys.argv[ 1 ]
  proorg_re = re.compile( r'proorg=(.*)' )

  proorg = proorg_re.findall( url )
  proorg = proorg[ 0 ]

  kb = urllib.urlopen( url )
  lines = kb.readlines()

  kb_f = open( proorg + '.flr', 'w' )

  for i in lines:
    kb_f.write( i )
  kb_f.close()
  kb.close()
  print proorg
else:
  raise Exception,'No url supplied!'
```

Using this script the following predicate could be implemented in $\mathcal{F}$LORA-*2* , loading any knowledge base from an URL.

```
loadKB( ?url ) :-
  str_cat( 'python get_kb.py ', ?url, ?cmd )@_prolog( string ),
  shell_to_list( ?cmd, [ [ ?kb ] ], ?_ )@_prolog( shell ),
  _add(?kb).
```

Now since the knowledge bases are now local, an agent would look similar to:

```
?- _add( loadKB ).

load_project :-
  loadKB('http://autopoiesis.foi.hr/flora2export.php?proorg=↘
  →Jupiter'),
  loadKB('http://autopoiesis.foi.hr/flora2export.php?proorg=↘
  →Saturn'),
```

```
  loadKB('http://autopoiesis.foi.hr/flora2export.php?proorg=↘
  →Neptun').

?- load_projects.

/* Agent definition ... */
```

Some of the presented queries are, off course, to complex to be learned and issued by normal users. This is why the next step in ⊤AOPīs is the development of easy-to-use querying mechanisms similar to query-by-example approaches and visual wiki search.

9.4 Other Examples of Possible Applications

Possible applications that came to mind by supporting autopoiesis in information systems range from generic purposes like project management to specialized applications like personal computer security. In the following we give a brief overview of ideas and concepts that will hopefully sparkle new ideas in the readers mind.

First of all there is the generic project management application for dynamic organizations. Organizations could by implementing a dynamic information system similar to the ⊤AOPīs system create a self-organizing project organization similar to the previously described fishnet organization or to create a dynamic project team layer above its normal (everyday) business structure like described in the section about the hypertext organization. By introducing adequate rules and procedures organization's members could perceive opportunities in the organization's environment more quickly transforming it into organizational projects.

The idea of open organizations [106] mostly concerned with public and political organizations is another possible example. Opening up the organization to the public that everyone who wants can join could yield better public perception of political processes. Public projects like a system for comparing different products and services according to price, quality etc. could yield more comprehensive results than professional web sites.

The notion of virtual organizations as well as joint ventures could also be supported through such systems. By establishing an autopoiesis oriented stock for cooperation between companies organizations will be likely to identify potential partners for

establishing virtual organizations or joint ventures through social networks.

E-learning is an example we showed to be successfully in the experiment we conducted since all the projects between students were conducted in a distributed environment. Students worked on their projects from home, school or any place they had Internet access.

Information system integration is another example. By using semantically defined web services and script extensions a group of people could merge different information systems into a more autopoiesis oriented one by organizing functionality they need.

All these applications, except e-learning, are only concepts that may or may not succeed since they haven't been implemented nor tested. Thus in order to test our premises we need to conduct future research and let time decide.

Chapter 10

Conclusion

The main aim of this thesis was to acknowledge that wiki systems operate in a complex environment - an autopoietic social system that surrounds them. The formalized knowledge that emerges on a wiki or semantic wiki system is the result of structural coupling of the social system to the very wiki interface. Due to the complexity of the social system which is sometime inconsistent with it self, it is certain that inconsistencies in formalized knowledge will emerge.

The idea of introducing semantic technologies into wiki system to facilitate the emergence of formalized knowledge seems promising, but it ignores the fact that most wiki systems are successful because of their ease of use. By introducing sophisticated semantic technologies the potential user base of a wiki system decreases to only those users that are or can become familiarized with such technologies.

Hence, this thesis provides two solutions to these problems: (1) acknowledging the possibility of inconsistencies and building mechanisms to deal with such; (2) hiding semantic technologies into the background of the system. The first solution resides on using social network analysis to provide a trust factor in any fact that is derived from a semantic wikis knowledge base. The second resides on using social tagging as a means to obtain semantic meta information.

Due to the Web 2.0 and more recently Web 3.0 paradigms as well as due to the popularity of such systems a lot of wiki systems with lots of interesting features were developed. Most of them feature their own syntax conventions usually denoted with wiki text or wiki syntax. After introducing wiki systems in chapter 2 we developed a formalization of wiki languages in chapter 3 using regular expressions. The set of introduced regular expressions can be seen as a hands on guide to implementing a wiki syntax parser.

The idea of the semantic web, a web of machine readable information that would allow for automated knowledge discovery by intelligent agents was further on described in chapter 4. By introducing an object oriented approach building upon [58] important ideas like domains, concepts, objects, relations, methods and attributes were introduced. Such a view of a particular domain was later on used to develop a formalization of semantic wiki systems building upon frame logic in chapter 5. We showed that any wiki page can be considered to be a generic object that contributors can shape to reflect their particular view. By using attribute-value tags, hyperlinks, and web services contributors can create attributes of an object (page), relations to other objects (pages) on the web as well as shape the behavior of the object (page) trough methods (services). By introducing special tags objects can be classified and class hierarchies can be build. Such user-obtained meta information was used to conceptualize the syntax of semantic wiki systems.

Autopoietic theory, a theory of complex, non-linear and especially living systems was then described in chapter 6. As of the initial definition given by Maturana and Varela [60] in biology, the theory found its way through the social sciences and formal organization theory introduced by Luhmann [53]. Still these two conceptualizations of autopoiesis (self-creation) are similar but in some cases incompatible and inconsistent. After developing a critique on autopoietic theory we gave guidelines for a new foundation that could be able to override these inconsistencies and allow the introduction of autopoiesis to the information sciences.

As it comes out, the main problem was to create a different conceptualization of autopoiesis in social and organizational systems that would introduce individuals, as opposed to communication and only communication proposed by Luhmann. Hence we conceptualized social systems as systems comprising of accepted individuals that accepted to be part of the social system. After introducing a genetic definition of information systems [13] we were able to show that information systems are in fact autopoietic since they overlap with the definition of social and organizational systems as defined by Luhmann. From our perspective autopoietic information systems are defined as sets of relations between communicative events that reproduce new communicative events based on previous (stored) communication. The organization of such systems (in Maturana's and Varela's sense) are the relations between communicative events described through their semantics (meaning). The structure of these systems (in Maturana's and Varela's sense) are the means that are used to produce communication described through syntax.

The acknowledgement that informations systems are autopoietic, as well as the fact that we can observe three types of information systems (societal, interactional and organizational) lets us conclude that autopoiesis in such systems can be facilitated by using adequate information technology. Technology is the environment that can be used to obtain formalized knowledge from an information system due to structural coupling processes. Wiki systems, and especially semantic wiki systems are one such technology. While "traditional' wiki systems are easy to use and thus acknowledge their complex environment, semantic wiki systems are complex and don't consider these facts.

In order to contribute to technology and systems that would be able to facilitate autopoiesis the concept of autopoiesis facilitating semantic wiki systems was considered in chapter 7. The main idea was to introduce social network analysis to obtain a fishnet structure [37] by creating dynamic hierarchies of contributors based on mutual trust relations. To formalize these hierarchies a variant of Bonacich's eigenvector centrality [11], the so called PageRank algorithm [12, 72] was used. This algorithm was especially convenient since the sum of all ranks of actors in a given social network equals to 1, reflecting thus a probability that a given actor will say the truth.

To connect social network analysis to semantic wiki languages an annotation scheme was proposed. Each meta data statement (given by some contributors through attribute - value tags, hyperlinks or web services) is annotated with a corresponding trust level. These levels can be used to filter inadequate results in some query. Due to the fact that we were able to formalize wiki languages, semantic wiki languages, as well as to introduce a fishnet structure and provide a probability annotation scheme, we confirm hypothesis 1.

In a semantic web environment intelligent agents have to gather information from distributed sources. The dynamic querying of such distributed knowledge repositories is known as amalgamation [51]. Due to the fact that the probability annotations of a given semantic wiki knowledge base reflect only the local social network, we needed to introduce an amalgamation scheme that will allow to annotate distributed sources. This was done trough knowledge base and social network integration, that allowed us to re-annotate the amalgamated knowledge bases. Thus hypothesis 2 is also confirmed.

In the end we formalized this new amalgamated and annotated semantic wiki language, described the semantic of queries in such a language and confirmed hypothesis 3. In chapter 8 we provided an implementation of such a language entitled **niKlas** after Niklas Luhmann who was an inspiration to this work. **niKlas** has been implemented

into the ᴛAOPīs system that provides a platform for self-organizing communities. Such communities can be either organizations or projects for which ᴛAOPīs provides suitable tools like semantic wiki systems, forums, blogs, ranking mechanisms, content filtering, and tagging facilities.

In chapter 9 we analyzed few possible application of such autopoiesis facilitating systems. The examples showed that such systems could be used in a wide range of applicative areas or better said in any situation where there is need for collaborative knowledge management of a group of people.

The most ungrateful task is to produce forecasts on a domain that is under intensive development like the one of autopoiesis in organizations and information systems presented in this thesis. Nevertheless we will try to give a short outline of situations that we find likely to occur.

Dynamic organizations will probably use dynamic web applications similar to the described ones to connect to customers and organizations in their environment. Since the world became networked the environment of today's organizations is almost everyone available via Internet. Through the use of such systems this fact will become more and more obvious.

We envision that only organizations that will recognize the need to support autopoiesis (that is decisions that allow for dynamic recreation of meaning) and develop adequate information systems for this task will be able to survive in the future. Rigid, bureaucratic, highly structured and non-adaptable organizational forms will probably die due to a turbulent environment that is to complex for such forms to process.

Organizations that want not only to survive but also to be successful should take modern organizational concepts, modern information and communication technologies as well as the autopoietic nature of information systems into consideration. An information system is an important subsystem of the organization that allows it to process and reduce complexity from its environment. If the processing capacity[1] of the information system isn't sufficient the organization will probably disappear back into the environment.

In the field of autopoiesis and information systems empirical studies of the applicability of autopoiesis to information systems have to be conducted. Additionally, formal methods to represent, model and predict autopoietic systems have to be developed. Such and other concerns are subject to future research of the author.

[1]Not to be misunderstood as the processing capacity of information and communication technologies that implement an information system.

Bibliography

[1] Abele, T., and Bischoff, V. Fraktal+: Adaptability in the age of e-business and networking,. In *Innovations for an e-Society* (2001), pp. 1–6.

[2] Abou-Zeid, E.-S. An autopoietic view of the concept 'information system'. In *Proceedings of the IFIP TC8/WG8. 1 International Conference on Information System Concepts: An Integrated Discipline Emerging* (2000), Kluwer, pp. 165–186.

[3] Auer, S., Dietzold, S., and Riechert, T. OntoWiki – A Tool for Social, Semantic Collaboration. In *The Semantic Web – ISWC 2006*. Springer-Verlag, Berlin, Germany, 2006, pp. 736–749.

[4] Auer, S., Jungmann, B., and Schonefeld, F. Semantic wiki representations for building an enterprise knowledge base. In *Reasoning Web. Third International Summer School 2007. Tutorial Lectures. (Lecture Notes in Computer Science vol. 4636)* (Berlin, Germany, 2007), Springer-Verlag, pp. 330–333.

[5] Baker, K., and Branch, K. *Concepts Underlying Organizational Effectiveness: Trends in the Organization and Management Science Literature*. Unpublished manuscript, 2002, pp. 1–14.

[6] Bao, J., Ding, L., Smart, P. R., Braines, D., and Jones, G. Rule modeling using semantic mediawiki. In *3rd Annual Conference of the International Technology Alliance (ACITA'09)* (Sept. 2009).

[7] Barnatt, C. Office space, cyberspace & virtual organization. *Journal of General Management 20*, 4 (1995), 78–91.

[8] Bača, M., Schatten, M., and Deranja, D. Autopoietic information systems in modern organizations. *Organizacija, Journal of Management, Informatics and Human Resources 40*, 3 (2007), 157–165.

[9] Belhajjame, K., Embury, S., Paton, N., Stevens, R., and Goble, A. Automatic annotations of semantic web services based on workflow definitions. *ACM Transactions on the Web*, 2 (April 2008), 1–34.

[10] Berners-Lee, T. The semantic web stack from a 2000 presentation. available at `http://www.w3.org/2000/Talks/1206-xml2k-tbl/`, accessed: 20th July 2008., 2000.

[11] Bonacich, P. Factoring and weighting approaches to clique identification. *Journal of Mathematical Sociology*, 2 (1972), 113–120.

[12] Brin, S., and Page, L. The anatomy of a large-scale hypertextual web search engine. In *Computer Networks and ISDN Systems* (1998), pp. 107–117.

[13] Brumec, J. A contribution to is general taxonomy. *Zbornik radova Fakulteta organizacije i informatike 21*, 1 (1997), 1–14.

[14] Brumec, J., and Vrček, N. Strategic planning of information systems (spis) – a survey of methodology. *Journal of Computing and Information Technology 10*, 3 (2002), 241—-247.

[15] Buble, M. *Management malog poduzeća*. Faculty of Economy Split, 2003.

[16] Buffa, M., Gandon, F., Ereteo, G., Sander, P., and Faron, C. SweetWiki: A semantic wiki. *JOURNAL OF WEB SEMANTICS 6*, 1 (FEB 2008), 84–97.

[17] c2.com. Wiki engines. Available at `http://c2.com/cgi/wiki?WikiEngines`, accessed: 2nd May 2009.

[18] Campanini, S. E., Castagna, P., and Tazzoli, R. Platypus wiki: a semantic wiki wiki web. In *Semantic Web Applications and Perspectives (SWAP) 1st Italian Semantic Web Workshop* (Ancona, Italy, 10th December 2004 2004).

[19] Cardoso, J., and Sheth, A. P. *Semantic Web Services, Processes and Applications*. Springer, New York, 2006.

[20] Churchill, C. Managing growth: The organizational architecture of microfinance institutions. In *USAID Microenterprise Best Practices Project Paper* (1997), pp. 7–26, 81–87.

[21] CTI. Can your organization survive a tsunami? Available at `http://www.ctiarch.com/oa/optassess1.htm`, 2004.

[22] Cunningham, W. Correspondence on the etymology of wiki. available at `http://c2.com/doc/etymology.html`, Accessed: 3rd September 2007, 2003.

[23] Cunningham, W. Wiki wiki hyper card. available at `http://c2.com/cgi/wiki?WikiWikiHyperCard`, Accessed: 3rd September 2007, 2003.

[24] De Paoli, F., and Loregian, M. Tools to foster semantic-based collaboration. a knowledge management approach based on a semantic wiki and personal ontologies. In *Third International Conference on Web information systems and technologies, WEBIST* (Setubal, Portugal, 2007), INSTICC, pp. 302–307.

[25] Di Iorio, A., Presutti, V., and Vitali, F. WikiFactory: An ontology-based application for creating domain-oriented wikis. In *SEMANTIC WEB: RESEARCH AND APPLICATIONS, PROCEEDINGS*, Sure, Y and Domingue, J, Ed., vol. 4011 of *LECTURE NOTES IN COMPUTER SCIENCE*. SPRINGER-VERLAG BERLIN, HEIDELBERGER PLATZ 3, D-14197 BERLIN, GERMANY, 2006, pp. 664–678.

[26] Diamond, G. March 2007 new words, oed. available at `http://dictionary.oed.com/news/newwords.html`. Accessed 16th March 2007, 2007.

[27] Divjak, B., and Lovrenčić, A. *Diskretna matematika s teorijom grafova*. TIVA & Faculty of Organization and Informatics, 2005.

[28] Dulčić, Ž.., Pavić, I., Rovan, M., and Veža, I. *Proizvodni menedžment*. Ekonomski fakultet Split, Fakultet of Electronics and Machinery and Shipbuilding Split, 1996.

[29] Dumitriu, S., Girdea, M., and Buraga, S. From Information Wiki to Knowledge Wiki via Semantic Web technologies. In *Innovations and Advanced Techniques in Computer and Information Sciences and Engineering*. Springer-Verlag, Berlin, Germany, 2007, pp. 443–448.

[30] Ebersbach, A. *Wiki: Web Collaboration*. Springer Science+Business Media, 2008.

[31] Fischer, J., Gantner, Z., Rendle, S., Stritt, M., and Schmidt-Thieme, L. Ideas and Improvements for Semantic Wikis. In *The Semantic Web: Research and Applications*. Springer-Verlag, Berlin, Germany, 2006, pp. 650–663.

[32] Galbraith, J., Downey, D., and Kates, A. *Designing Dynamic Organizations*. AMACOM, 2001.

[33] Haase, P., Herzig, D., Musen, M., and Tran, T. Semantic Wiki Search. In *The Semantic Web: Research and Applications*. Springer-Verlag, Berlin, Germany, 2009, pp. 445–460.

[34] Hoehndorf, R., Bacher, J., Backhaus, M., Gregorio, J. S. E., Loebe, F., Pruefer, K., Uciteli, A., Visagie, J., Herre, H., and Kelso, J. BOWiki: an ontology-based wiki for annotation of data and integration of knowledge in biology. *BMC BIOINFORMATICS 10*, Suppl. 5 (MAY 6 2009).

[35] Huner, K. M., and Otto, B. The effect of using a semantic wiki for metadata management: a controlled experiment. In *42nd Hawaii International Conference on System Sciences. HICSS-42* (Piscataway, NJ, USA, 2009), IEEE, p. 9.

[36] Jinhyun, A., Jung, J., and Key-Sun, C. Interleaving ontology mapping for online semantic annotation on semantic wiki. In *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology - Workshops* (Piscataway, NJ, USA, 2008), vol. 3, IEEE, pp. 25–28.

[37] Johansen, R., and Swigart, R. *Upsizing The Individual In The Downsized Corporation Managing In The Wake Of Reengineering, Globalization, And Overwhelming Technological Change*. Perseus Publishing, 2000.

[38] Kawamoto, K., Kitamura, Y., and Tijerino, Y. Kawawiki: a semantic wiki based on rdf templates. In *IEEE/WIC/ACM International Conference on Web Intelligence International Intelligence Agent Technology Workshops* (Los Alamitos, CA, USA, 2006), IEEE Comput. Soc., p. 5.

[39] Kawamoto, K., Mase, M., Kitamura, Y., and Tijerino, Y. Semantic wiki where human and agents collaborate. In *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology - Workshops* (Piscataway, NJ, USA, 2008), vol. 3, IEEE, pp. 147–151.

[40] Kiesel, M., and van Elst L Buscher G Schwarz, S. Mymory: enhancing a semantic wiki with context annotations. In *Semantic Web: Research and Applications. 5th European Semantic Web Conference, ESWC 2008* (Berlin, Germany, 2008), Springer-Verlag, pp. 817–821.

[41] Kifer, M., Lausen, G., and Wu, J. Logical foundations of object-oriented and frame-based languages. *Journal of the Association for Computing Machinery 42* (May 1995), 741–843.

[42] Kim, H., and Choi, J. A semantic web-enabled wiki system for ontology construction and sharing. *Journal of KISS: Software and Applications 33*, 8 (2006), 703–717.

[43] Kotis, K. On supporting hcome-30 ontology argumentation using semantic wiki technology. In *On the Move to Meaningful Internet Systems: OTM 2008 Workshops.OTM Confederated International Workshops and Posters ADI, AWeSoMe, COMBEK, EON, IWSSA, MONET, OnToContent+QSI, ORM, PerSys, RDDS, SEMELS, and SWWS* (Berlin, Germany, 2008), Springer-Verlag, pp. 193–199.

[44] Krakar, Z. Upravljanje informatizacijom. Faculty of Organization and Informatics, Available at `http://www.foi.hr/CMS_library/studiji/dodiplomski/IS/kolegiji/uis/skripta3.zip`, 2004.

[45] Krötzsch, M., Schaffert, S., and Vrandecic, D. Reasoning in semantic wikis. In *Reasoning Web Summer School* (Dresden, September 2007), Springer-Verlag.

[46] Krötzsch, M., and Vrandecic, D. Semantic Wikipedia. In *Social Semantic Web*. Springer-Verlag, Berlin, Germany, 2009, pp. 393–421.

[47] Lange, C. Swim: a semantic wiki for mathematical knowledge management. In *Semantic Web: Research and Applications. 5th European Semantic Web Conference, ESWC 2008* (Berlin, Germany, 2008), Springer-Verlag, pp. 832–837.

[48] Lau, A. S. M. Implementation of an onto-wiki toolkit using web services to improve the efficiency and effectiveness of medical ontology co-authoring and analysis. *INFORMATICS FOR HEALTH & SOCIAL CARE 34*, 1 (2009), 73–80.

[49] Leuf, B., and Cunningham, W. *The Wiki Way. Quick collaboration on the Web*. Addison-Wesley, 2001.

[50] Lovrenčić, A. *Logički programski jezici za izgradnju sustava za integriranje heterogenih izvora znanja (Logical Programming Languages for the Development of Heterogenous Knowledge Sources Integration Systems)*. Ph.d. diss., Faculty of Organization and Informatics, Varaždin, 2003.

[51] Lovrenčić, A., and Čubrilo, M. Amalgamation of heterogeneous data sources using amalgamated annotated hilog. In *Proceedings of 3rd IEEE Conference on Intelligent Engineering Systems, INES'99* (1999).

[52] Lu, Q., ying Jiao, Y., and Chen, J. Uimwiki: an enhanced semantic wiki for user information management. In *IEEE International Symposium on IT in Medicine and Education (ITME)* (Piscataway, NJ, USA, 2008), IEEE, pp. 930–934.

[53] Luhmann, N. *Soziale Systeme: Grundriß einer allgemeinen Theorie*. Suhrkamp, Frankfurt, Germany, 1984.

[54] Luhmann, N. Law as a social system. *Northwestern University Law Review 83* (1989), 136–150.

[55] Luhmann, N. *Observations on modernity*. Stanford University Press, Stanford, 1998.

[56] Luhmann, N. Organization. In *Autopoietic Organization Theory Drawing on Niklas Luhmann's Social Systems Perspective*, T. Bakken and T. Hernes, Eds. Abstract, Liber, Copenhagen Business School Press, Oslo, 2003, pp. 31–53.

[57] Maleković, M., and Schatten, M. Leadership in team based knowledge management - an autopoietic information system's perspective. In *19th Central European Conference on Information and Intelligent Systems – CECIIS2008 Conference Proceedings* (September 2008), B. Aurer and M. Bača, Eds., Faculty of Organization and Informatics, pp. 47–52.

[58] Martin, J., and Odell, J. J. *Object-Oriented Methods: A Foundation*, uml edition ed. Prentice Hall PTR, New Jersey, 1998.

[59] Maturana, H. R. The organization of the living: A theory of the living organization. *International Journal of Man-Machine Studies 7* (1975), 313–332.

[60] Maturana, H. R., and Varela, F. J. Autopoiesis: The organization of the living. In *Autopoiesis and cognition*, H. R. Maturana and F. J. Varela, Eds. Reidel, Boston, 1973, pp. 59–138.

[61] May, W. *How to Write F-Logic Programs in Florid - A Tutorial for the Database Language F-Logic - Version 3.0 (FloXML)*, Oct. 2000. Available at `http://dbis.`

informatik.uni-freiburg.de/content/projects/Florid/florid_tutorial.pdf, accessed 27nd September 2009.

[62] McIlraith, S. A., Son, T. C., and Zeng, H. Semantic web services. *IEEE Intelligent Systems 16*, 2 (2001), 46–53.

[63] Miciunas, G. Cre/fm organizational architecture: Structuring staff success. The Environments Group, Available at http://www.envgroup.com/browse/presentations/IFMA2002_staffsuccesspaper.pdf, 2002.

[64] Mika, P. *Social Networks and the Semantic Web*. Springer, New York, 2007.

[65] Mingers, J. Observing organizations: An evaluation of luhmann's organization theory. In *Autopoietic Organization Theory Drawing on Niklas Luhmann's Social Systems Perspective*, T. Bakken and T. Hernes, Eds. Abstract, Liber, Copenhagen Business School Press, Oslo, 2003, pp. 103–122.

[66] Mitsuhiko, T. Autopoiesis of information systems and network society. *Joho Shori Gakkai Kenkyu Hokoku IS-89*, 88 (2004), 1–8.

[67] Muljadi, H., Takeda, H., Shakya, A., Kawamoto, S., Kobayashi, S., Fujiyama, A., and Ando, K. Semantic wiki as a lightweight knowledge management system. In *The Semantic Web - ASWC 2006. First Asian Semantic Web Conference. Proceedings (Lecture Notes in Computer Science Vol. 4185)* (Berlin, Germany, 2006), Springer-Verlag, pp. 65–71.

[68] Nadler, D. A., Gerstein, M. S., and Shaw, R. B. *Organizational Architecture, Designs for Changing Organizations*. Jossey-Bass, San Francisco, 1992.

[69] Nonaka, I., and Takeuchi, H. *The Knowledge-Creating Company, How Japanese Companies Create the Dynamics of Innovation*. Oxford University Press, 1995.

[70] oddmuse.org. What is a wiki? Available at http://www.oddmuse.org/cgi-bin/oddmuse/What_Is_A_Wiki, accessed 2nd May 2009.

[71] Oren, E., Völkel, M., Breslin, J. G., and Decker, S. Semantic Wikis for Personal Knowledge Management. In *Database and Expert Systems Applications*. Springer-Verlag, Berlin, Germany, 2006, pp. 509–518.

[72] Page, L., Brin, S., Motwani, R., and Winograd, T. The pagerank citation ranking: Bringing order to the web, 1999.

[73] Pansanato, L. T. E., and Fortes, R. P. M. System Description: An Orienteering Strategy to Browse Semantically-Enhanced Educational Wiki Pages. In *The Semantic Web: Research and Applications*. Springer-Verlag, Berlin, Germany, 2007, pp. 809–818.

[74] Prud'hommeaux, E., and Seaborne, A. Sparql query language for rdf, w3c recommendation. Tech. rep., World Wide Web Consortium, Jan. 2008.

[75] Quick, T. Autopoiesis. `http://www.cs.ucl.ac.uk/staff/t.quick/autopoiesis.html`, (accessed: 01-04-2008), 2003.

[76] Rado, D. Führung und organisation (leadership and organization). Universität St. Gallen – Hochschule für Wirtschafts- , Rechts- und Sozialwissenschaften, Available at `http://www.mavericks.ch/3download/_files/_unisg/unisg_sum_FuO_ws0102_v10.pdf`, 2002.

[77] Raymond, E. S. The cathedral and the bazaar. Available at `http://www.tuxedo.org/~esr/`, 2005.

[78] Reihlen, M., and Rohde, A. Das heterarchische unternehmen: ein flexibiles organisationsmodell im wissensbasierten wettbewerb (the heterarchic business: a flexible organizational model in a knowledge-based competition environment). *Zeitschrift Lernende Organisation 8* (2002), 30–34.

[79] Rhee, S. K., Lee, J., and Park, M.-W. RIKI: A Wiki-Based Knowledge Sharing System for Collaborative Research Projects. In *Computer-Human Interaction*. Springer-Verlag, Berlin, Germany, 2008, pp. 68–76.

[80] SalzburgResearch. Ikewiki. available at `http://ikewiki.salzburgresearch.at/`, Accessed: 14th May 2008.

[81] Schaffert, S. Ikewiki: A semantic wiki for collaborative knowledge management. In *1st International Workshop on Semantic Technologies in Collaborative Applications STICA 06 Proceedings* (Manchester, UK, 2006).

[82] Schaffert, S., Bry, F., Baumeister, J., and Kiesel, M. Semantic wiki. *Informatik-Spektrum 30*, 6 (2007), 434–439.

[83] Schaffert, S., Eder, J., Grünwald, S., Kurz, T., and Radulescu, M. KiWi – A Platform for Semantic Social Software (Demonstration). In *The Semantic Web: Research and Applications*. Springer-Verlag, Berlin, Germany, 2009, pp. 888–892.

[84] Schaffert, S., Westenthaler, R., and Gruber, A. Ikewiki: A user-friendly semantic wiki. In *Demos and Posters of the 3rd European Semantic Web Conference (ESWC 2006)* (Budva, Montenegro, 11th – 14th June 2006).

[85] Schatten, M., Brumec, J., and Višić, M. Strategic planning of an autopoietic information system. In *Proceedings of the IIS 2007 18th International Conferece on Information and Intelligent Systems* (2007), B. Aurer and M. Bača, Eds., Faculty of Organization and Informatics, pp. 435–440.

[86] Schatten, M., Maleković, M., and Rabuzin, K. Inconsistencies in semantic social web applications. In *Proceedings of the 20th Central European Conference on Information and Intelligent Systems* (2009), B. Aurer, M. Bača, and K. Rabuzin, Eds., Faculty of Organization and Informatics.

[87] Schatten, M., Čubrilo, M., and Ševa, J. A semantic wiki system based on f-logic. In *19th Central European Conference on Information and Intelligent Systems – CECIIS2008 Conference Proceedings* (2008), B. Aurer and M. Bača, Eds., Faculty of Organization and Informatics, pp. 57–61.

[88] Schatten, M., and Žugaj, M. Organizing a fishnet structure. In *29th International Conference Information Technology Interfaces Proceedings* (Cavtat-Dubrovnik, Croatia, June 25 — 28 2007), pp. 81–86.

[89] Semantic-mediawiki.org. Semantic mediawiki. available at `http://semantic-mediawiki.org/`, Accessed: 14th May 2008.

[90] Siles, A., Lopez-Cima, A., Corcho, O., and Gomez-Perez, A. Odewiki: a semantic wiki that interoperates with the odesew semantic portal. In *Semantic Web: Research and Applications. 5th European Semantic Web Conference, ESWC 2008* (Berlin, Germany, 2008), Springer-Verlag, pp. 859–863.

[91] Silverman, L. L. Organizational architecture: A framework for successful transformation. 1997.

[92] Singh, A. V., Wombacher, A., and Aberer, K. Personalized Information Access in a Wiki Using Structured Tagging. In *On the Move to Meaningful Internet Systems 2007: OTM 2007 Workshops*. Springer-Verlag, Berlin, Germany, 2007, pp. 427–436.

[93] Slonneger, K., and Kurtz, B. L. *Formal Syntax and Semantics of Programming Languages – A Laboratory Based Approach*. Addison-Wesley Publishing Company, 1995.

[94] Smith, G. *Tagging: People-Powered Metadata for the Social Web*. New Riders, Berkeley, 2008.

[95] Souzis, A. Building a semantic wiki. *IEEE Intelligent Systems 20*, 5 (2005), 87–91.

[96] Stalbaum, B. Toward autopoietic database. Research report for C5 corporation, available at `http://www.c5corp.com/research/autopoieticdatabase.shtml`, accessed: 18th August 2007.

[97] Sung-Kooc, L., and In-Young, K. Collaborative ontology construction using template-based wiki for semantic web applications. In *International Conference on Computer Engineering and Technology. ICCET 2009* (Piscataway, NJ, USA, 2009), vol. 2, IEEE, pp. 171–175.

[98] SweetWiki. Senamtic web enabled technology wiki. available at `http://argentera.inria.fr/wiki/data/Main/MainHome.jsp`, Accessed: 14th May 2008.

[99] Valenzuela, C. Y. Dentro de la selección (within selection). *Revista Chilena de Historia Natural 80*, 1 (2007), 109–116.

[100] van der Blonk, H., Huysman, M., and Spoor, E. Autopoiesis and the evolution of information systems. Tech. rep., 1998.

[101] van Elst, L., Kiesel, M., Schwarz, S., Buscher, G., Lauer, A., and Dengel, A. Contextualized knowledge acquisition in a personal semantic wiki. In *Knowledge Engineering: Practice and Patterns. 16th International Conference, EKAW 2008* (Berlin, Germany, 2008), Springer-Verlag, pp. 172–187.

[102] Varela, F. J. *Principles of Biological Autonomy*. Elsevier, New York, North Holland, 1979.

[103] Völkel, M., Krötzsch, M., Vrandecic, D., Haller, H., and Studer, R. Semantic wikipedia. In *International World Wide Web Conference (IW3C2), WWW 2006 Proceedings* (Edinburgh, Scotland, May 23—26 2006).

[104] Žugaj, M., Dumičić, K., and Dušak, V. *Temelji znanstvenoistraživačkog rada - Metodologija i metodika*. TIVA & Faculty of Organization and Informatics, Varaždin, 2006.

[105] Žugaj, M., and Schatten, M. *Arhitektura suvremenih organizacija*. Tonimir and Faculty of Organization and Informatics, Varaždinske Toplice, Croatia, 2005.

[106] Žugaj, M., and Schatten, M. Nekoliko riječi o otvorenoj organizaciji. In *5th International Conference on Production Engineering, RIM 2005., Scientific Book* (Bihać, Bosnia and Herzegowina, 14 - 17 September 2005), pp. 917–922.

[107] Žugaj, M., and Schatten, M. Otvorena ontologija organizacijske arhitekture u funkciji upravljanja znanjem. *Ekonomski vjesnik XX*, 1 – 2 (2007), 39–45.

[108] Žugaj, M., Šehanović, J., and Cingula, M. *Organizacija*. TIVA & Faculty of Organization and Informatics, Varaždin, Croatia, 2004.

[109] W3C. Owl web ontology language overview - w3c recommendation, Feb. 2004.

[110] Wang, C., Lu, J., Zhang, G., and Zeng, X. Creating and managing ontology data on the web: a semantic wiki approach. In *Web Information Systems Engineering - WISE 2007. Proceedings 8th International Conference on Web Information Systems Engineering. (Lecture Notes in Computer Science vol. 4831)* (Berlin, Germany, 2007), Springer-Verlag, pp. 513–522.

[111] Warnecke, H.-J. Die fraktale fabrik - produzieren im netzwerk (the fractal company - production in the network). In *GI Jahrestagung* (1992), pp. 20–33.

[112] Wasserman, S., and Faust, K. *Social Network Analysis ; Methods and Applications*. Structural analysis in the social sciences. Cambridge University Press, 1994.

[113] Whitaker, R. Tutorial 1: Introductory orientation. `http://www.enolagaia.com/Tutorial1.html`, (accessed: 01-12-2005), 2001.

[114] Whitaker, R. Tutorial 2: Concepts and constructs. `http://www.enolagaia.com/Tutorial2.html`, (accessed: 20-03-2008), 2001.

[115] Wikipedia. History of wikis — wikipedia, the free encyclopedia. available at `http://en.wikipedia.org/w/index.php?title=History_of_wikis&oldid=293627468`, accessed 1st June 2009., 2009.

[116] Yang, G., Kifer, M., and Zhao, C. Flora-2: A rule-based knowledge representation and inference infrastructure for the semantic web. In *Second International Conference on Ontologies, Databases and Applications of Semantics (ODBASE) Proceedings* (Catania, Sicily, Italy, November 2003).

Appendix A

Wiki Parser for niKlas in XSB Prolog

The presented BNF grammar (chapter 3) was implemented in XSB Prolog to automatically construct parse trees as shown in the following listing.

```
:- import shell/1 from shell.
:- import str_cat/2,
          term_to_atom/2 from string.
:- import re_match/5,
          re_substring/4 from regmatch.
:- import member/2 from basics.

:- auto_table.

% lexical structure

statement( text( X ) ) -->
  [ X ], { any_string_not_in_keywords( X ) }.

internal_url( internal_url( X ) ) -->
  [ X ],
  {
    any_string_not_in_keywords( X ),
    not( match_url( X ) )
  }.

external_url( external_url( X ) ) -->
```

```
  [ X ], { match_url( X ) }.

link_name( link_name( X ) ) -->
  [ X ], { any_string_not_in_keywords( X ) }.

reference_citation_name( reference_citation_name( X ) ) -->
  [ X ], { any_string_not_in_keywords( X ) }.

formatting_expression_start(
  formatting_expression_start( '[h1]' ) ) -->
    [ '[h1]' ].

formatting_expression_end(
  formatting_expression_end( '[/h1]' ) ) -->
    [ '[/h1]' ].

formatting_expression_start(
  formatting_expression_start( '[h2]' ) ) -->
    [ '[h2]' ].

formatting_expression_end(
  formatting_expression_end( '[/h2]' ) ) -->
    [ '[/h2]' ].

formatting_expression_start(
  formatting_expression_start( '[h3]' ) ) -->
    [ '[h3]' ].

formatting_expression_end(
  formatting_expression_end( '[/h3]' ) ) -->
    [ '[/h3]' ].

formatting_expression_start(
  formatting_expression_start( '[j]' ) ) -->
```

```
    [ '[j]' ].

formatting_expression_end(
  formatting_expression_end( '[/j]' ) ) -->
    [ '[/j]' ].

formatting_expression_start(
  formatting_expression_start( '[b]' ) ) -->
    [ '[b]' ].

formatting_expression_end(
  formatting_expression_end( '[/b]' ) ) -->
    [ '[/b]' ].

formatting_expression_start(
  formatting_expression_start( '[i]' ) ) -->
    [ '[i]' ].

formatting_expression_end(
  formatting_expression_end( '[/i]' ) ) -->
    [ '[/i]' ].

formatting_expression_start(
  formatting_expression_start( '[center]' ) ) -->
    [ '[center]' ].

formatting_expression_end(
  formatting_expression_end( '[/center]' ) ) -->
    [ '[/center]' ].

formatting_expression_start(
  formatting_expression_start( '[code]' ) ) -->
    [ '[code]' ].
```

```
formatting_expression_end(
  formatting_expression_end( '[/code]' ) ) -->
    [ '[/code]' ].

formatting_expression_start(
  formatting_expression_start( '[quote]' ) ) -->
    [ '[quote]' ].

formatting_expression_end(
  formatting_expression_end( '[/quote]' ) ) -->
    [ '[/quote]' ].

display_object_start(
  display_object_start( '[tube]' ) ) -->
    [ '[tube]' ].

display_object_end(
  display_object_end( '[/tube]' ) ) -->
    [ '[/tube]' ].

image_start(
  image_start( '[img=' ) ) -->
    [ '[img=' ].

image_end(
  image_end( ']' ) ) -->
    [ ']' ].



comment_start(
  comment_start( '[comment]' ) ) -->
    [ '[comment]' ].

comment_end(
```

```
  comment_end( '[/comment]' ) ) -->
    [ '[/comment]' ].

hyperlink_start(
  hyperlink_start( '[link=' ) ) -->
    [ '[link=' ].

hyperlink_end(
  hyperlink_end( ']' ) ) -->
    [ ']' ].

variable_template(
  variable_template( '[ouline]' ) ) -->
    [ '[outline]' ].

reference_entry_begin(
  reference_entry_begin( '[ref=' ) ) -->
    [ '[ref=' ].

reference_entry_end(
  reference_entry_end( ']' ) ) -->
    [ ']' ].

cite_key(
  cite_key( X ) ) -->
    [ X ],
    { any_string_not_in_keywords( X ) }.

reference_citation_start(
  reference_citation_start( '[cite=' ) ) -->
    [ '[cite=' ].

reference_citation_end(
  reference_citation_end( ']' ) ) -->
```

```
    [ ']' ].

wtable_start(
  wtable_start( '[table]' ) ) -->
    [ '[table]' ].

wtable_end(
  wtable_end( '[/table]' ) ) -->
    [ '[/table]' ].

% regex & aux

corresponding_tags( X, Y ) :-
  re_match(
    '\[(.*)\]',
    X,
    0,
    -,
    [
      match( _, _ ),
      match( B1, E1 )
    ]
  ),
  re_match(
    '\[[/](.*)\]',
    Y,
    0,
    -,
    [
      match( _, _ ),
      match( B2, E2 )
    ]
  ),
  re_substring( X, B1, E1, R1 ),
```

```
  re_substring( Y, B2, E2, R2 ),
  R1 = R2.

any_string_not_in_keywords( X ) :-
  Keywords = [ '[h1]', '[/h1]', '[h2]', '[/h2]', '[h3]', '[/h3]↘
  →', '[j]', '[/j]', '[b]', '[/b]', '[i]', '[/i]', '[center]', ↘
  →'[/center]', '[code]', '[/code]', '[quote]', '[/quote]', '[↘
  →tube]', '[/tube]', '[img=', ']', '[comment]', '[/comment]', ↘
  →'[link=', '[url]', '[/url]', '[ref', '[cite', '[table]', '[/↘
  →table]', '&&', '##' ],
  not(
    member( X, Keywords )
  ).

match_url( X ) :-
  re_match(
    '[a-zA-Z]+://([.]?[a-zA-Z0-9_/-])*',
    X,
    0,
    -,
    L
  ).

% wiki grammar

wiki_page(
  wiki_page( X ) ) -->
    statements( X ).

statements(
  statements( [ X ] ) ) -->
    statement( X ).

statements(
```

```
  statements( [ X | Y ] ) ) -->
    statement( X ),
    statements( statements( Y ) ).

statement(
  statement( E ) ) -->
    formatting_expression( E ).

statement(
  statement( D ) ) -->
    display_object( D ).

statement(
  statement( I ) ) -->
    image( I ).

statement(
  statement( C ) ) -->
    comment( C ).

statement(
  statement( L ) ) -->
    hyperlink( L ).

statement(
  statement( V ) ) -->
    variable_template( V ).

statement(
  statement( R ) ) -->
    reference_entry( R ).

statement(
  statement( C ) ) -->
```

```
    reference_citation( C ).

statement(
  statement( T ) ) -->
    wtable( T ).

formatting_expression(
  formatting_expression( Start, Statement, End ) ) -->
    formatting_expression_start( Start ),
    statements( Statement ),
    formatting_expression_end( End ),
    {
      Start = formatting_expression_start( S ),
      End = formatting_expression_end( E ),
      corresponding_tags( S, E )
    }.

display_object(
  display_object( Start, URL, End ) ) -->
    display_object_start( Start ),
    external_url( URL ),
    display_object_end( End ).

display_object(
  display_object( Start, Text, End ) ) -->
    display_object_start( Start ),
    statement( Text ),
    display_object_end( End ).

image(
  image( Start, URL, End ) ) -->
    image_start( Start ),
    external_url( URL ),
    image_end( End ).
```

```
comment(
  comment( Start, S, End ) ) -->
    comment_start( Start ),
    statements( S ),
    comment_end( End ).

hyperlink(
  hyperlink( Start, URL, Name, End ) ) -->
    hyperlink_start( Start ),
    internal_url( URL ),
    [ '>' ],
    link_name( Name ),
    hyperlink_end( End ).

hyperlink(
  hyperlink( Start, URL, Name, End ) ) -->
    hyperlink_start( Start ),
    external_url( URL ),
    [ '>' ],
    link_name( Name ),
    hyperlink_end( End ).

reference_entry(
  reference_entry( Begin, CiteKey, End ) ) -->
    reference_entry_begin( Begin ),
    cite_key( CiteKey ),
    reference_entry_end( End ).

reference_citation(
  reference_citation( Start, CiteKey, Name, End ) ) -->
    reference_citation_start( Start ),
    cite_key( CiteKey ),
    [ '>' ],
```

```
    reference_citation_name( Name ),
    reference_citation_end( End ).

wtable(
  wtable( Start, Rows, End ) ) -->
    wtable_start( Start ),
    wtable_rows( Rows ),
    wtable_end( End ).

wtable_rows(
  wtable_rows( [ Row ] ) ) -->
    wtable_row( Row ).

wtable_rows(
  wtable_rows( [ Row | Rows ] ) ) -->
    wtable_row( Row ),
    [ '##' ],
    wtable_rows(
      wtable_rows( Rows )
    ).

wtable_row(
  wtable_row( Cells ) ) -->
    wtable_cells( Cells ).

wtable_cells(
  wtable_cells( [ Cell ] ) ) -->
    wtable_cell( Cell ).

wtable_cells(
  wtable_cells( [ Cell | Cells ] ) ) -->
    wtable_cell( Cell ),
    [ '&&' ],
    wtable_cells(
```

```
      wtable_cells( Cells )
    ).

wtable_cell(
  wtable_cell( Statements ) ) -->
    statements( Statements ).

% parsing and auxiliary

pprint( Term ) :-
  term_to_atom( Term, Atom ),
  str_cat( 'python pprint.py ''', Atom, Part ),
  str_cat( Part, '''', Command ),
  shell( Command ).

read_code( List ) :-
  shell_to_list( 'python readcode.py', [ List ], _ ).

parse :-
  read_code( X ),
  wiki_page( T, X, [] ),
  pprint( T ).

parse_t :-
  read_code( X ),
  trace,
  wiki_page( T, X, [] ),
  notrace,
  pprint( T ).

?- parse.
```

Appendix B

Semantic Wiki Parser for niKlas in XSB Prolog

The presented BNF grammar (chapter 5) was implemented in XSB Prolog to automatically construct parse trees as shown in the following listing.

```
:- import shell/1 from shell.
:- import str_cat/2,
          term_to_atom/2 from string.
:- import re_match/5,
          re_substring/4 from regmatch.
:- import member/2 from basics.

:- auto_table.

% lexical structure

statement(
  text( X ) ) -->
    [ X ],
    {
      any_string_not_in_keywords( X ),
      not( fl_var( X ) )
    }.

statement(
```

```
  text(
    flogic_variable( X ) ) ) -->
      [ X ],
      { fl_var( X ) }.

internal_url(
  internal_url( X ) ) -->
    [ X ],
    {
      any_string_not_in_keywords( X ),
      not( url( X ) ),
      not( fl_var( X ) )
    }.

internal_url(
  internal_url(
    flogic_variable( X ) ) ) -->
      [ X ],
      { fl_var( X ) }.

external_url(
  external_url( X ) ) -->
    [ X ],
    {
      url( X ),
      not( fl_var( X ) )
    }.

external_url(
  external_url(
    flogic_variable( X ) ) ) -->
      [ X ],
      { fl_var( X ) }.
```

```
link_name(
  link_name( X ) ) -->
    [ X ],
    {
      any_string_not_in_keywords( X ),
      not( fl_var( X ) )
    }.

link_name(
  link_name(
    flogic_variable( X ) ) ) -->
      [ X ],
      { fl_var( X ) }.

reference_citation_name(
  reference_citation_name( X ) ) -->
    [ X ],
    {
      any_string_not_in_keywords( X ),
      not( fl_var( X ) )
    }.

reference_citation_name(
  reference_citation_name(
    flogic_variable( X ) ) ) -->
      [ X ],
      { fl_var( X ) }.

formatting_expression_start(
  formatting_expression_start( '[h1]' ) ) -->
    [ '[h1]' ].

formatting_expression_end(
  formatting_expression_end( '[/h1]' ) ) -->
```

```
    [ '[/h1]' ].

formatting_expression_start(
  formatting_expression_start( '[h2]' ) ) -->
    [ '[h2]' ].

formatting_expression_end(
  formatting_expression_end( '[/h2]' ) ) -->
    [ '[/h2]' ].

formatting_expression_start(
  formatting_expression_start( '[h3]' ) ) -->
    [ '[h3]' ].

formatting_expression_end(
  formatting_expression_end( '[/h3]' ) ) -->
    [ '[/h3]' ].

formatting_expression_start(
  formatting_expression_start( '[j]' ) ) -->
    [ '[j]' ].

formatting_expression_end(
  formatting_expression_end( '[/j]' ) ) -->
    [ '[/j]' ].

formatting_expression_start(
  formatting_expression_start( '[b]' ) ) -->
    [ '[b]' ].

formatting_expression_end(
  formatting_expression_end( '[/b]' ) ) -->
    [ '[/b]' ].
```

```
formatting_expression_start(
  formatting_expression_start( '[i]' ) ) -->
    [ '[i]' ].

formatting_expression_end(
  formatting_expression_end( '[/i]' ) ) -->
    [ '[/i]' ].

formatting_expression_start(
  formatting_expression_start( '[center]' ) ) -->
    [ '[center]' ].

formatting_expression_end(
  formatting_expression_end( '[/center]' ) ) -->
    [ '[/center]' ].

formatting_expression_start(
  formatting_expression_start( '[code]' ) ) -->
    [ '[code]' ].

formatting_expression_end(
  formatting_expression_end( '[/code]' ) ) -->
    [ '[/code]' ].

formatting_expression_start(
  formatting_expression_start( '[quote]' ) ) -->
    [ '[quote]' ].

formatting_expression_end(
  formatting_expression_end( '[/quote]' ) ) -->
    [ '[/quote]' ].

formatting_expression_start(
  formatting_expression_start( '[header]' ) ) -->
```

```
    [ '[header]' ].

formatting_expression_end(
  formatting_expression_end( '[/header]' ) ) -->
    [ '[/header]' ].



display_object_start(
  display_object_start( '[tube]' ) ) -->
    [ '[tube]' ].

display_object_end(
  display_object_end( '[/tube]' ) ) -->
    [ '[/tube]' ].

image_start(
  image_start( '[img=' ) ) -->
    [ '[img=' ].

image_end(
  image_end( ']' ) ) -->
    [ ']' ].

comment_start(
  comment_start( '[comment]' ) ) -->
    [ '[comment]' ].

comment_end(
  comment_end( '[/comment]' ) ) -->
    [ '[/comment]' ].

hyperlink_start(
  hyperlink_start( '[link=' ) ) -->
    [ '[link=' ].
```

```
hyperlink_end(
  hyperlink_end( ']' ) ) -->
    [ ']' ].

variable_template(
  variable_template( '[ouline]' ) ) -->
    [ '[outline]' ].

reference_entry_begin(
  reference_entry_begin( '[ref=' ) ) -->
    [ '[ref=' ].

reference_entry_end(
  reference_entry_end( ']' ) ) -->
    [ ']' ].

cite_key(
  cite_key( X ) ) -->
    [ X ],
    { any_string_not_in_keywords( X ) }.

reference_citation_start(
  reference_citation_start( '[cite=' ) ) -->
    [ '[cite=' ].

reference_citation_end(
  reference_citation_end( ']' ) ) -->
    [ ']' ].

wtable_start(
  wtable_start( '[table]' ) ) -->
    [ '[table]' ].
```

```
wtable_end(
  wtable_end( '[/table]' ) ) -->
    [ '[/table]' ].

query_start(
  query_start( '[query=', FLQuery ) ) -->
    [ '[query=' ],
    flogic_query( FLQuery ),
    [ ']' ].

query_end(
  query_end( '[/query]' ) ) -->
    [ '[/query]' ].

flogic_isa_symbol(
  flogic_isa_symbol( ':' ) ) -->
    [ ':' ].

flogic_isa_symbol(
  flogic_isa_symbol( '::' ) ) -->
    [ '::' ].

flogic_implication_symbol(
  flogic_implication_symbol( ':-' ) ) -->
    [ ':-' ].

flogic_query_symbol(
  flogic_query_symbol( '?-' ) ) -->
    [ '?-' ].

flogic_method_arrow1(
  flogic_method_arrow1( '->' ) ) -->
    [ '->' ].
```

```
flogic_method_arrow1(
  flogic_method_arrow1( '*->' ) ) -->
    [ '*->' ].

flogic_method_arrow2(
  flogic_method_arrow2( '=>' ) ) -->
    [ '=>' ].

flogic_method_arrow2(
  flogic_method_arrow2( '*=>' ) ) -->
    [ '*=>' ].

flogic_dot(
  flogic_dot( '.' ) ) -->
    [ '.' ].

flogic_dot(
  flogic_dot( '..' ) ) -->
    [ '..' ].

flogic_dot(
  flogic_dot( '!' ) ) -->
    [ '!' ].

flogic_predicate(
  flogic_predicate( X ) ) -->
    [ X ],
    { any_string( X ) }.

flogic_predicate(
  flogic_predicate(
    flogic_variable( X ) ) ) -->
      [ X ],
      { fl_var( X ) }. % HiLog extension
```

```
flogic_build_in_predicate(
  flogic_build_in_predicate( X ) ) -->
    [ X ],
    { fl_string( X ) }.

flogic_build_in_infix_predicate(
  flogic_build_in_infix_predicate( '<' ) ) -->
    [ '<' ].

flogic_build_in_infix_predicate(
  flogic_build_in_infix_predicate( '>' ) ) -->
    [ '>' ].

flogic_build_in_infix_predicate(
  flogic_build_in_infix_predicate( '=' ) ) -->
    [ '=' ].

flogic_build_in_infix_predicate(
  flogic_build_in_infix_predicate( '=<' ) ) -->
    [ '=<' ].

flogic_build_in_infix_predicate(
  flogic_build_in_infix_predicate( '>=' ) ) -->
    [ '>=' ].

flogic_build_in_operator(
  flogic_build_in_operator( '+' ) ) -->
    [ '+' ].

flogic_build_in_operator(
  flogic_build_in_operator( '-' ) ) -->
    [ '-' ].
```

```
flogic_build_in_operator(
  flogic_build_in_operator( '*' ) ) -->
    [ '*' ].

flogic_build_in_operator(
  flogic_build_in_operator( '/' ) ) -->
    [ '/' ].

flogic_functor(
  flogic_functor( X ) ) -->
    [ X ],
    { any_string( X ) }.

flogic_functor(
  flogic_functor(
    flogic_variable( X ) ) ) -->
      [ X ],
      { fl_var( X ) }. % HiLog extension

flogic_variable(
  flogic_variable( X ) ) -->
    [ X ],
    { fl_var( X ) }.

flogic_string(
  flogic_string( X ) ) -->
    [ X ],
    { any_string( X ) }.

flogic_integer(
  flogic_integer( X ) ) -->
    [ X ],
    { any_integer( X ) }.
```

```
% regex & aux

corresponding_tags( X, Y ) :-
  re_match(
    '\[(.*)\]',
    X,
    0,
    -,
    [ match( _, _ ), match( B1, E1 ) ]
  ),
  re_match(
    '\[[/](.*)\]',
    Y,
    0,
    -,
    [ match( _, _ ), match( B2, E2 ) ]
  ),
  re_substring( X, B1, E1, R1 ),
  re_substring( Y, B2, E2, R2 ),
  R1 = R2.

any_string_not_in_keywords( X ) :-
  Keywords = [ '[h1]', '[/h1]', '[h2]', '[/h2]', '[h3]', '[/h3]↘
  →', '[j]', '[/j]', '[b]', '[/b]', '[i]', '[/i]', '[center]', ↘
  →'[/center]', '[code]', '[/code]', '[quote]', '[/quote]', '[↘
  →tube]', '[/tube]', '[img=', ']', '[comment]', '[/comment]', ↘
  →'[link=', '[url]', '[/url]', '[ref', '[cite', '[table]', '[/↘
  →table]', '&&', '##', '[query=', '[/query]', '[header]', '[/↘
  →header]' ],
  not( member( X, Keywords ) ).

url( X ) :-
  not( number( X ) ),
  re_match(
```

```
    '^[a-zA-Z]+://([.]?[a-zA-Z0-9_/-])*$',
    X,
    0,
    -,
    L
  ).

any_string( X ) :-
  not( number( X ) ),
  re_match(
    "^([a-zA-Z0-9_]*)$|^(\'.*\')$",
    X,
    0,
    -,
    L
  ).

fl_var( X ) :-
  not( number( X ) ),
  re_match(
    "^[?][a-zA-Z0-9_]*$",
    X,
    0,
    -,
    L
  ).

fl_string( X ) :-
  not( number( X ) ),
  re_match(
    "^fl[a-zA-Z_]*$",
    X,
    0,
    -,
```

```
    L
  ).

any_integer( X ) :-
  not( number( X ) ),
  re_match( '^[0-9]*$', X, 0, _, L ).


any_integer( X ) :-
  integer( X ).


% wiki grammar

semantic_wiki_page(
  semantic_wiki_page( X ) ) -->
    statements( X ).

semantic_wiki_page(
  semantic_wiki_page( X, Y ) ) -->
    statements( X ),
    [ '|||--o0o--|||' ],
    meta_info( Y ).

statements(
  statements( [ X ] ) ) -->
    statement( X ).

statements(
  statements( [ X | Y ] ) ) -->
    statement( X ),
    statements( statements( Y ) ).

statement(
```

```
  statement( E ) ) -->
    formatting_expression( E ).

statement(
  statement( D ) ) -->
    display_object( D ).

statement(
  statement( I ) ) -->
    image( I ).

statement(
  statement( C ) ) -->
    comment( C ).

statement(
  statement( L ) ) -->
    hyperlink( L ).

statement(
  statement( V ) ) -->
    variable_template( V ).

statement(
  statement( R ) ) -->
    reference_entry( R ).

statement(
  statement( C ) ) -->
    reference_citation( C ).

statement(
  statement( T ) ) -->
    wtable( T ).
```

```
statement(
  statement( Q ) ) -->
    query( Q ).

formatting_expression(
  formatting_expression( Start, Statement, End ) ) -->
    formatting_expression_start( Start ),
    statements( Statement ),
    formatting_expression_end( End ),
    {
      Start = formatting_expression_start( S ),
      End = formatting_expression_end( E ),
      corresponding_tags( S, E )
    }.

display_object(
  display_object( Start, URL, End ) ) -->
    display_object_start( Start ),
    external_url( URL ),
    display_object_end( End ).

display_object(
  display_object( Start, Text, End ) ) -->
    display_object_start( Start ),
    statement( Text ),
    display_object_end( End ).

image(
  image( Start, URL, End ) ) -->
    image_start( Start ),
    external_url( URL ),
    image_end( End ).
```

```
comment(
  comment( Start, S, End ) ) -->
    comment_start( Start ),
    statements( S ),
    comment_end( End ).

hyperlink(
  hyperlink( Start, URL, Name, End ) ) -->
    hyperlink_start( Start ),
    internal_url( URL ),
    [ '>' ],
    link_name( Name ),
    hyperlink_end( End ).

hyperlink(
  hyperlink( Start, URL, Name, End ) ) -->
    hyperlink_start( Start ),
    external_url( URL ),
    [ '>' ],
    link_name( Name ),
    hyperlink_end( End ).

reference_entry(
  reference_entry( Begin, CiteKey, End ) ) -->
    reference_entry_begin( Begin ),
    cite_key( CiteKey ),
    reference_entry_end( End ).

reference_citation(
  reference_citation( Start, CiteKey, Name, End ) ) -->
    reference_citation_start( Start ),
    cite_key( CiteKey ), [ '>' ],
    reference_citation_name( Name ),
    reference_citation_end( End ).
```

```
wtable(
  wtable( Start, Rows, End ) ) -->
    wtable_start( Start ),
    wtable_rows( Rows ),
    wtable_end( End ).

wtable_rows(
  wtable_rows( [ Row ] ) ) -->
    wtable_row( Row ).

wtable_rows(
  wtable_rows( [ Row | Rows ] ) ) -->
    wtable_row( Row ),
    [ '##' ],
    wtable_rows( wtable_rows( Rows ) ).

wtable_row(
  wtable_row( Cells ) ) -->
    wtable_cells( Cells ).

wtable_cells(
  wtable_cells( [ Cell ] ) ) -->
    wtable_cell( Cell ).

wtable_cells(
  wtable_cells( [ Cell | Cells ] ) ) -->
    wtable_cell( Cell ),
    [ '&&' ],
    wtable_cells( wtable_cells( Cells ) ).

wtable_cell(
  wtable_cell( Statements ) ) -->
    statements( Statements ).
```

```
% query and frame logic grammar

query(
  query( Start, STem, End ) ) -->
    query_start( Start ),
    semantic_template( STem ),
    query_end( End ).

semantic_template(
  semantic_template( VarStat ) ) -->
    statements( VarStat ).

flogic_query(
  flogic_query( X ) ) -->
    flogic_rule_body( X ), [ '.' ].

flogic_rule_head(
  flogic_rule_head( X ) ) -->
    flogic_list_of_molecules( X ).

flogic_rule_body(
  flogic_rule_body( B ) ) -->
    flogic_list_of_literals( B ).

flogic_list_of_molecules(
  flogic_list_of_molecules( [ X ] ) ) -->
    flogic_molecule( X ).

flogic_list_of_molecules(
  flogic_list_of_molecules( [ X | Y ] ) ) -->
    flogic_molecule( X ),
    [ ',' ],
    flogic_list_of_molecules( flogic_list_of_molecules( Y ) ).
```

```
flogic_list_of_literals(
  flogic_list_of_literals( [ X ] ) ) -->
    flogic_literal( X ).

flogic_list_of_literals(
  flogic_list_of_literals( [ X | Y ] ) ) -->
    flogic_literal( X ),
    [ ',' ],
    flogic_list_of_literals( flogic_list_of_literals( Y ) ).

flogic_literal(
  flogic_literal( 'not', X ) ) -->
  [ 'not' ],
  flogic_molecule( X ).

flogic_literal(
  flogic_literal( 'not', X ) ) -->
    [ 'not' ],
    [ '(' ],
    flogic_molecule( X ),
    [ ')' ].

flogic_literal(
  flogic_literal( X ) ) -->
    flogic_molecule( X ).

flogic_molecule(
  flogic_molecule( X ) ) -->
    flogic_fmolecule( X ).

flogic_molecule(
  flogic_molecule( X ) ) -->
    flogic_pmolecule( X ).
```

```
flogic_pmolecule(
  flogic_pmolecule( P ) ) -->
    flogic_predicate( P ).

flogic_pmolecule(
  flogic_pmolecule( P, L ) ) -->
    flogic_predicate( P ),
    [ '(' ],
    flogic_list_of_expressions( L ),
    [ ')' ].

flogic_pmolecule(
  flogic_pmolecule( B ) ) -->
    flogic_build_in_predicate( B ).

flogic_pmolecule(
  flogic_pmolecule( B, L ) ) -->
    flogic_build_in_predicate( B ),
    [ '(' ],
    flogic_list_of_expressions( L ),
    [ ')' ].

flogic_pmolecule(
  flogic_pmolecule( A1, O, A2 ) ) -->
    flogic_arithmetic_expression( A1 ),
    flogic_build_in_infix_predicate( O ),
    flogic_arithmetic_expression( A2 ).

flogic_list_of_expressions(
  flogic_list_of_expressions( [ E ] ) ) -->
    flogic_expression( E ).

flogic_list_of_expressions(
```

```
  flogic_list_of_expressions( [ E | R ] ) -->
    flogic_expression( E ),
    [ ',' ],
    flogic_list_of_expressions( flogic_list_of_expressions( R )↘
    → ).

flogic_expression(
  flogic_expression( P ) ) -->
    flogic_path_expression( P ).

flogic_expression(
  flogic_expression( F ) ) -->
    flogic_fmolecule( F ).

flogic_expression(
  flogic_expression( A ) ) -->
    flogic_aggregrate( A ).

flogic_arithmetic_expression(
  flogic_arithmetic_expression( E ) ) -->
    flogic_expression( E ).

flogic_arithmetic_expression(
  flogic_arithmetic_expression( A1, O, A2 ) ) -->
    flogic_arithmetic_expression( A1 ),
    flogic_build_in_operator( O ),
    flogic_arithmetic_expression( A2 ).

flogic_arithmetic_expression(
  flogic_arithmetic_expression( E ) ) -->
    [ '(' ],
    flogic_arithmetic_expression( E ),
    [ ')' ].
```

```
flogic_aggregrate(
  flogic_aggregrate( Agg, AV, Q ) ) -->
    flogic_id_term( Agg ),
    [ '{' ],
    flogic_variable( AV ),
    [ '|' ],
    flogic_list_of_literals ,
    [ '}' ].

flogic_aggregrate(
  flogic_aggregrate( Agg, AV, GV, Q ) ) -->
    flogic_id_term( Agg ),
    [ '{' ],
    flogic_variable( AV ),
    [ '[' ],
    flogic_list_of_variables( GV ),
    [ ']' ],
    [ '|' ],
    flogic_list_of_literals , [ '}' ].

flogic_fmolecule(
  flogic_fmolecule( P, S ) ) -->
    flogic_path_expression( P ),
    flogic_specification( S ).

flogic_path_expression(
  flogic_path_expression( Id ) ) -->
    flogic_id_term( Id ).

flogic_path_expression(
  flogic_path_expression( E ) ) -->
    [ '(' ],
    flogic_expression( E ), [ ')' ].
```

```
flogic_path_expression(
  flogic_path_expression( E, D, M ) ) -->
    flogic_path_expression( E ),
    flogic_dot( D ),
    flogic_method_application( M ).

flogic_path_expression(
  flogic_path_expression( F, D, M ) ) -->
    flogic_fmolecule( F ),
    flogic_dot( D ),
    flogic_method_application( M ).

flogic_specification(
  flogic_specification( I ) ) -->
    flogic_isa_specification( I ).

flogic_specification(
  flogic_specification( I ) ) -->
    flogic_isa_specification( I ),
    [ '[' ],
    [ ']' ].

flogic_specification(
  flogic_specification( I, M ) ) -->
    flogic_isa_specification( I ),
    [ '[' ],
    flogic_list_of_methods( M ),
    [ ']' ].

flogic_specification(
  flogic_specification( [] ) ) -->
    [ '[' ],
    [ ']' ].
```

```
flogic_specification(
  flogic_specification( M ) ) -->
    [ '[' ],
    flogic_list_of_methods( M ),
    [ ']' ].

flogic_isa_specification(
  flogic_isa_specification( S, I ) ) -->
    flogic_isa_symbol( S ),
    flogic_id_term( I ).

flogic_isa_specification(
  flogic_isa_specification( S, E ) ) -->
    flogic_isa_symbol( S ),
    [ '(' ],
    flogic_expression( E ),
    [ ')' ].

flogic_method_application(
  flogic_method_application( I ) ) -->
    flogic_id_term( I ).

flogic_method_application(
  flogic_method_application( I, L ) ) -->
    flogic_id_term( I ),
    [ '(' ],
    flogic_list_of_expressions( L ),
    [ ')' ]. % flora2 doesn't use the '@' sign to separate ↘
    →methods from their arguments since with HiLog extensions ↘
    →it became redundant (floraManual, p. 14)

flogic_method_application(
  flogic_method_application( E ) ) -->
    [ '(' ],
```

```
    flogic_expression( E ),
    [ ')' ].

flogic_method_application(
  flogic_method_application( E, L ) ) -->
    [ '(' ],
    flogic_expression( E ),
    [ ')' ],
    [ '(' ],
    flogic_list_of_expressions( L ),
    [ ')' ].

flogic_list_of_methods(
  flogic_list_of_methods( [ MS ] ) ) -->
    flogic_method_specification( MS ).

flogic_list_of_methods(
  flogic_list_of_methods( [ MS | L ] ) ) -->
    flogic_method_specification( MS ),
    [ ',' ],
    flogic_list_of_methods( flogic_list_of_methods( L ) ).

flogic_method_specification(
  flogic_method_specification( Ma, Mr ) ) -->
    flogic_method_application( Ma ),
    flogic_method_result( Mr ).

flogic_method_specification(
  flogic_method_specification( 'not', Ma, Mr ) ) -->
  [ 'not' ],
  [ '(' ],
  flogic_method_application( Ma ),
  flogic_method_result( Mr ),
  [ ')' ].
```

```
flogic_method_result(
  flogic_method_result( Arr, E ) ) -->
    flogic_method_arrow1( Arr ),
    flogic_expression( E ).

flogic_method_result(
  flogic_method_result( Arr, E ) ) -->
    flogic_method_arrow1( Arr ),
    [ '{' ],
    flogic_list_of_expressions( E ),
    [ '}' ].

flogic_method_result(
  flogic_method_result( Arr, E ) ) -->
  flogic_method_arrow2( Arr ),
  flogic_expression( E ).

flogic_method_result(
  flogic_method_result( Arr, E ) ) -->
    flogic_method_arrow2( Arr ),
    [ '{' ],
    flogic_list_of_expressions( E ),
    [ '}' ].

flogic_id_term(
  flogic_id_term( B ) ) -->
    flogic_basic_id_term( B ).

flogic_id_term(
  flogic_id_term( F, E ) ) -->
    flogic_functor( F ),
    [ '(' ],
    flogic_list_of_expressions( E ),
```

```
    [ ')' ].

flogic_basic_id_term(
  flogic_basic_id_term( F ) ) -->
    flogic_functor( F ).

flogic_basic_id_term(
  flogic_basic_id_term( V ) ) -->
    flogic_variable( V ).

flogic_basic_id_term(
  flogic_basic_id_term( S ) ) -->
    flogic_string( S ).

flogic_basic_id_term(
  flogic_basic_id_term( I ) ) -->
    flogic_integer( I ).

flogic_list_of_variables(
  flogic_list_of_variables( [ X ] ) ) -->
    flogic_variable( X ).

flogic_list_of_variables(
  flogic_list_of_variables( [ X | Y ] ) ) -->
    flogic_variable( X ),
    [ ',' ],
    flogic_list_of_variables( flogic_list_of_variables( Y ) ).

% meta info grammar

meta_info(
  meta_info( [ T ] ) ) -->
    attribute_value_tag( T ).
```

```
meta_info(
  meta_info( [ L ] ) ) -->
    hyperlink( L ).

meta_info(
  meta_info( [ T | R ] ) ) -->
    attribute_value_tag( T ),
    meta_info( meta_info( R ) ).

meta_info(
  meta_info( [ L | R ] ) ) -->
    hyperlink( L ),
    meta_info( meta_info( R ) ).

attribute_value_tag(
  attribute_value_tag( A, V ) ) -->
    attribute( A ),
    [ ':' ],
    value( V ).

attribute_value_tag(
  attribute_value_tag( class, V ) ) -->
    [ class ],
    [ ':' ],
    flogic_id_term( V ).

attribute_value_tag(
  attribute_value_tag( subclass, V ) ) -->
    [ subclass ],
    [ ':' ],
    flogic_id_term( V ).

attribute_value_tag(
  attribute_value_tag( subclass, E ) ) -->
```

```
    [ subclass ],
    [ ':' ],
    [ '(' ],
    flogic_expression( E ),
    [ ')' ].

attribute_value_tag(
  attribute_value_tag( rule, H, I, B ) ) -->
    [ rule ],
    [ ':' ],
    flogic_rule_head( H ),
    flogic_implication_symbol( I ),
    flogic_rule_body( B ).

attribute(
  attribute( A ) ) -->
    flogic_id_term( A ).

value(
  value( V ) ) -->
    flogic_id_term( V ).

internal_url(
  internal_url( U ) ) -->
    flogic_id_term( U ).

external_url(
  external_url( U ) ) -->
    flogic_id_term( U ).

% parsing & auxilliary predicates

pprint( Term ) :-
  term_to_atom( Term, Atom ),
```

```
  str_cat( 'python pprint.py ''', Atom, Part ),
  str_cat( Part, '''', Command ),
  shell( Command ).

read_code( List ) :-
  shell_to_list( 'python readcode.py', [ List ], _ ).

parse :-
  read_code( X ),
  semantic_wiki_page( T, X, [] ),
  pprint( T ).

parse_t :-
  read_code( X ),
  trace,
  semantic_wiki_page( T, X, [] ),
  notrace,
  pprint( T ).

?- parse.
```

Appendix C

Amalgamated Annotated Semantic Wiki Parser for niKlas in XSB Prolog

The presented BNF grammar (chapter 7) was implemented in XSB Prolog to automatically construct parse trees as shown in the following listing.

```
:- import shell/1 from shell.
:- import str_cat/2,
          term_to_atom/2 from string.
:- import re_match/5,
          re_substring/4 from regmatch.
:- import member/2 from basics.
:- op( 100, yfx, '@' ).

:- auto_table.

% lexical structure

statement(
  text( X ) ) -->
    [ X ],
    {
      any_string_not_in_keywords( X ),
      not( fl_var( X ) )
    }.
```

```
statement(
  text( flogic_variable( X ) ) ) -->
    [ X ],
    { fl_var( X ) }.

internal_url(
  internal_url( X ) ) -->
    [ X ],
    {
      any_string_not_in_keywords( X ),
      not( url( X ) ),
      not( fl_var( X ) )
    }.

internal_url(
  internal_url( flogic_variable( X ) ) ) -->
    [ X ],
    { fl_var( X ) }.

external_url(
  external_url( X ) ) -->
    [ X ],
    {
      url( X ),
      not( fl_var( X ) )
    }.

external_url(
  external_url( flogic_variable( X ) ) ) -->
    [ X ],
    { fl_var( X ) }.

link_name(
  link_name( X ) ) -->
```

```
    [ X ],
    {
      any_string_not_in_keywords( X ),
      not( fl_var( X ) )
    }.

link_name(
  link_name( flogic_variable( X ) ) -->
    [ X ],
    { fl_var( X ) }.

reference_citation_name(
  reference_citation_name( X ) ) -->
    [ X ],
    {
      any_string_not_in_keywords( X ),
      not( fl_var( X ) )
    }.

reference_citation_name(
  reference_citation_name(
    flogic_variable( X ) ) ) -->
    [ X ],
    { fl_var( X ) }.

formatting_expression_start(
  formatting_expression_start( '[h1]' ) ) -->
    [ '[h1]' ].

formatting_expression_end(
  formatting_expression_end( '[/h1]' ) ) -->
    [ '[/h1]' ].

formatting_expression_start(
```

```
  formatting_expression_start( '[h2]' ) ) -->
    [ '[h2]' ].

formatting_expression_end(
  formatting_expression_end( '[/h2]' ) ) -->
    [ '[/h2]' ].

formatting_expression_start(
  formatting_expression_start( '[h3]' ) ) -->
    [ '[h3]' ].

formatting_expression_end(
  formatting_expression_end( '[/h3]' ) ) -->
    [ '[/h3]' ].

formatting_expression_start(
  formatting_expression_start( '[j]' ) ) -->
    [ '[j]' ].

formatting_expression_end(
  formatting_expression_end( '[/j]' ) ) -->
    [ '[/j]' ].

formatting_expression_start(
  formatting_expression_start( '[b]' ) ) -->
    [ '[b]' ].

formatting_expression_end(
  formatting_expression_end( '[/b]' ) ) -->
    [ '[/b]' ].

formatting_expression_start(
  formatting_expression_start( '[i]' ) ) -->
    [ '[i]' ].
```

```
formatting_expression_end(
  formatting_expression_end( '[/i]' ) ) -->
    [ '[/i]' ].

formatting_expression_start(
  formatting_expression_start( '[center]' ) ) -->
    [ '[center]' ].

formatting_expression_end(
  formatting_expression_end( '[/center]' ) ) -->
    [ '[/center]' ].

formatting_expression_start(
  formatting_expression_start( '[code]' ) ) -->
    [ '[code]' ].

formatting_expression_end(
  formatting_expression_end( '[/code]' ) ) -->
    [ '[/code]' ].

formatting_expression_start(
  formatting_expression_start( '[quote]' ) ) -->
    [ '[quote]' ].

formatting_expression_end(
  formatting_expression_end( '[/quote]' ) ) -->
    [ '[/quote]' ].

formatting_expression_start(
  formatting_expression_start( '[header]' ) ) -->
    [ '[header]' ].

formatting_expression_end(
```

```
  formatting_expression_end( '[/header]' ) ) -->
    [ '[/header]' ].

display_object_start(
  display_object_start( '[tube]' ) ) -->
    [ '[tube]' ].

display_object_end(
  display_object_end( '[/tube]' ) ) -->
    [ '[/tube]' ].

image_start(
  image_start( '[img=' ) ) -->
    [ '[img=' ].

image_end(
  image_end( ']' ) ) -->
    [ ']' ].

comment_start(
  comment_start( '[comment]' ) ) -->
    [ '[comment]' ].

comment_end(
  comment_end( '[/comment]' ) ) -->
    [ '[/comment]' ].

hyperlink_start(
  hyperlink_start( '[link=' ) ) -->
    [ '[link=' ].

hyperlink_end(
  hyperlink_end( ']' ) ) -->
    [ ']' ].
```

```
variable_template(
  variable_template( '[ouline]' ) ) -->
    [ '[outline]' ].

reference_entry_begin(
  reference_entry_begin( '[ref=' ) ) -->
    [ '[ref=' ].

reference_entry_end(
  reference_entry_end( ']' ) ) -->
    [ ']' ].

cite_key(
  cite_key( X ) ) --> [ X ],
    { any_string_not_in_keywords( X ) }.

reference_citation_start(
  reference_citation_start( '[cite=' ) ) -->
    [ '[cite=' ].

reference_citation_end(
  reference_citation_end( ']' ) ) -->
    [ ']' ].

wtable_start(
  wtable_start( '[table]' ) ) -->
    [ '[table]' ].

wtable_end(
  wtable_end( '[/table]' ) ) -->
    [ '[/table]' ].

query_start(
```

```
  query_start( '[query=', FLQuery ) ) -->
    [ '[query=' ], flogic_query( FLQuery ), [ ']' ].

query_end(
  query_end( '[/query]' ) ) -->
    [ '[/query]' ].

flogic_isa_symbol(
  flogic_isa_symbol( ':' ) ) -->
    [ ':' ].

flogic_isa_symbol(
  flogic_isa_symbol( '::' ) ) -->
    [ '::' ].

flogic_implication_symbol(
  flogic_implication_symbol( ':-' ) ) -->
    [ ':-' ].

flogic_query_symbol(
  flogic_query_symbol( '?-' ) ) -->
    [ '?-' ].

flogic_method_arrow1(
  flogic_method_arrow1( '->' ) ) -->
    [ '->' ].

flogic_method_arrow1(
  flogic_method_arrow1( '*->' ) ) -->
    [ '*->' ].

flogic_method_arrow2(
  flogic_method_arrow2( '=>' ) ) -->
    [ '=>' ].
```

```
flogic_method_arrow2(
  flogic_method_arrow2( '*=>' ) ) -->
    [ '*=>' ].

flogic_dot(
  flogic_dot( '.' ) ) -->
    [ '.' ].

flogic_dot(
  flogic_dot( '..' ) ) -->
    [ '..' ].

flogic_dot(
  flogic_dot( '!' ) ) -->
    [ '!' ].

flogic_predicate(
  flogic_predicate( X ) ) -->
    [ X ],
    { any_string( X ) }.

flogic_predicate(
  flogic_predicate( flogic_variable( X ) ) ) -->
    [ X ],
    { fl_var( X ) }. % HiLog extension



flogic_build_in_predicate(
  flogic_build_in_predicate( X ) ) -->
    [ X ],
    { fl_string( X ) }.

flogic_build_in_infix_predicate(
```

```
  flogic_build_in_infix_predicate( '<' ) ) -->
    [ '<' ].

flogic_build_in_infix_predicate(
  flogic_build_in_infix_predicate( '>' ) ) -->
    [ '>' ].

flogic_build_in_infix_predicate(
  flogic_build_in_infix_predicate( '=' ) ) -->
    [ '=' ].

flogic_build_in_infix_predicate(
  flogic_build_in_infix_predicate( '=<' ) ) -->
    [ '=<' ].

flogic_build_in_infix_predicate(
  flogic_build_in_infix_predicate( '>=' ) ) -->
    [ '>=' ].

flogic_build_in_operator(
  flogic_build_in_operator( '+' ) ) -->
    [ '+' ].

flogic_build_in_operator(
  flogic_build_in_operator( '-' ) ) -->
    [ '-' ].

flogic_build_in_operator(
  flogic_build_in_operator( '*' ) ) -->
    [ '*' ].

flogic_build_in_operator(
  flogic_build_in_operator( '/' ) ) -->
    [ '/' ].
```

```
flogic_functor(
  flogic_functor( X ) ) -->
    [ X ],
    { any_string( X ) }.

flogic_functor(
  flogic_functor( flogic_variable( X ) ) ) -->
    [ X ],
    { fl_var( X ) }. % HiLog extension

flogic_variable(
  flogic_variable( X ) ) -->
    [ X ],
    { fl_var( X ) }.

flogic_string(
  flogic_string( X ) ) -->
    [ X ],
    { any_string( X ) }.

flogic_integer(
  flogic_integer( X ) ) -->
    [ X ],
    { any_integer( X ) }.

probability_statement_start(
  probability_statement_start( '[probability=' ) ) -->
    [ '[probability=' ].

probability_statement_end(
  probability_statement_end( ']' ) ) -->
    [ ']' ].
```

```
probability_value(
  probability_value( X ) ) -->
    [ X ],
    { X > 0, X < 1 }.

amalgamation_statement_start(
  amalgamation_statement_start( '[amalgamate=' ) ) -->
    [ '[amalgamate=' ].

amalgamation_statement_end(
  amalgamation_statement_end( ']' ) ) -->
    [ ']' ].

knowledge_base(
  knowledge_base( X ) ) -->
    [ X ],
    { any_string( X ) }.

% regex & aux

corresponding_tags( X, Y ) :-
  re_match(
    '\[(.*)\]',
    X,
    0,
    -,
    [ match( _, _ ), match( B1, E1 ) ]
  ),
  re_match(
    '\[[/](.*)\]',
    Y,
    0,
    -,
    [ match( _, _ ), match( B2, E2 ) ]
```

```
  ),
  re_substring( X, B1, E1, R1 ),
  re_substring( Y, B2, E2, R2 ),
  R1 = R2.

any_string_not_in_keywords( X ) :-
  Keywords = [ '[h1]', '[/h1]', '[h2]', '[/h2]', '[h3]', '[/h3]↘
  →', '[j]', '[/j]', '[b]', '[/b]', '[i]', '[/i]', '[center]', ↘
  →'[/center]', '[code]', '[/code]', '[quote]', '[/quote]', '[↘
  →tube]', '[/tube]', '[img=', ']', '[comment]', '[/comment]', ↘
  →'[link=', '[url]', '[/url]', '[ref', '[cite', '[table]', '[/↘
  →table]', '&&', '##', '[query=', '[/query]', '[header]', '[/↘
  →header]' ],
  not( member( X, Keywords ) ).

url( X ) :-
  not( number( X ) ),
  re_match(
    '^[a-zA-Z]+://([.]?[a-zA-Z0-9_/-])*$',
    X,
    0,
    -,
    L
  ).

any_string( X ) :-
  not( number( X ) ),
  re_match(
    "^([a-zA-Z0-9_]*)$|^(\'.*\')$",
    X,
    0,
    -,
    L
  ).
```

```
fl_var( X ) :-
  not( number( X ) ),
  re_match(
    "^[?][a-zA-Z0-9_]*$",
    X,
    0,
    -,
    L
  ).

fl_string( X ) :-
  not( number( X ) ),
  re_match(
    "^fl[a-zA-Z_]*$",
    X,
    0,
    -,
    L
  ).

any_integer( X ) :-
  not( number( X ) ),
  re_match(
    '^[0-9]*$',
    X,
    0,
    -,
    L
  ).



any_integer( X ) :-
  integer( X ).
```

```
% wiki grammar

autopoiesis_facilitating_semantic_wiki_page(
  autopoiesis_facilitating_semantic_wiki_page( X ) ) -->
    statements( X ).

autopoiesis_facilitating_semantic_wiki_page(
  autopoiesis_facilitating_semantic_wiki_page( X, Y ) ) -->
    statements( X ),
    [ '|||--o0o--|||' ],
    meta_info( Y ).

statements(
  statements( [ X ] ) ) -->
    statement( X ).

statements(
  statements( [ X | Y ] ) ) -->
    statement( X ),
    statements( statements( Y ) ).

statement(
  statement( E ) ) -->
    formatting_expression( E ).

statement(
  statement( D ) ) -->
    display_object( D ).

statement(
  statement( I ) ) -->
    image( I ).
```

```
statement(
  statement( C ) ) -->
    comment( C ).

statement(
  statement( L ) ) -->
    hyperlink( L ).

statement(
  statement( V ) ) -->
    variable_template( V ).

statement(
  statement( R ) ) -->
    reference_entry( R ).

statement(
  statement( C ) ) -->
    reference_citation( C ).

statement(
  statement( T ) ) -->
    wtable( T ).

statement(
  statement( Q ) ) -->
    query( Q ).

formatting_expression(
  formatting_expression( Start, Statement, End ) ) -->
    formatting_expression_start( Start ),
    statements( Statement ),
    formatting_expression_end( End ),
```

```
    {
      Start = formatting_expression_start( S ),
      End = formatting_expression_end( E ),
      corresponding_tags( S, E )
    }.

display_object(
  display_object( Start, URL, End ) ) -->
    display_object_start( Start ),
    external_url( URL ),
    display_object_end( End ).

display_object(
  display_object( Start, Text, End ) ) -->
    display_object_start( Start ),
    statement( Text ),
    display_object_end( End ).

image(
  image( Start, URL, End ) ) -->
    image_start( Start ),
    external_url( URL ),
    image_end( End ).

comment(
  comment( Start, S, End ) ) -->
    comment_start( Start ),
    statements( S ),
    comment_end( End ).

hyperlink(
  hyperlink( Start, URL, Name, End ) ) -->
    hyperlink_start( Start ),
    internal_url( URL ),
```

```
    [ '>' ],
    link_name( Name ),
    hyperlink_end( End ).

hyperlink(
  hyperlink( Start, URL, Name, End ) ) -->
    hyperlink_start( Start ),
    external_url( URL ),
    [ '>' ],
    link_name( Name ),
    hyperlink_end( End ).

reference_entry(
  reference_entry( Begin, CiteKey, End ) ) -->
    reference_entry_begin( Begin ),
    cite_key( CiteKey ),
    reference_entry_end( End ).

reference_citation(
  reference_citation( Start, CiteKey, Name, End ) ) -->
    reference_citation_start( Start ),
    cite_key( CiteKey ),
    [ '>' ],
    reference_citation_name( Name ),
    reference_citation_end( End ).

wtable(
  wtable( Start, Rows, End ) ) -->
    wtable_start( Start ),
    wtable_rows( Rows ),
    wtable_end( End ).

wtable_rows(
  wtable_rows( [ Row ] ) ) -->
```

```
    wtable_row( Row ).

wtable_rows(
  wtable_rows( [ Row | Rows ] ) ) -->
    wtable_row( Row ),
    [ '##' ],
    wtable_rows( wtable_rows( Rows ) ).

wtable_row(
  wtable_row( Cells ) ) -->
    wtable_cells( Cells ).

wtable_cells(
  wtable_cells( [ Cell ] ) ) -->
    wtable_cell( Cell ).

wtable_cells(
  wtable_cells( [ Cell | Cells ] ) ) -->
    wtable_cell( Cell ),
    [ '&&' ],
    wtable_cells( wtable_cells( Cells ) ).

wtable_cell(
  wtable_cell( Statements ) ) -->
    statements( Statements ).

% query and frame logic grammar

query(
  query( Start, STem, End ) ) -->
    query_start( Start ),
    semantic_template( STem ),
    query_end( End ).
```

```
query(
  query( Start, PS, STem, End ) ) -->
    query_start( Start ),
    probability_statement( PS ),
    semantic_template( STem ),
    query_end( End ).

query(
  query( Start, PS, AmS, STem, End ) ) -->
    query_start( Start ),
      probability_statement( PS ),
      amalgamation_statement( AmS ),
      semantic_template( STem ),
      query_end( End ).

query(
  query( Start, AmS, STem, End ) ) -->
    query_start( Start ),
    amalgamation_statement( AmS ),
    semantic_template( STem ),
    query_end( End ).

semantic_template(
  semantic_template( VarStat ) ) -->
    statements( VarStat ).

probability_statement(
  probability_statement( Start, Op, Val, End ) ) -->
    probability_statement_start( Start ),
      flogic_build_in_infix_predicate( Op ),
      probability_value( Val ),
      probability_statement_end( End ).

amalgamation_statement(
```

```
  amalgamation_statement( Start, KB, End ) ) -->
    amalgamation_statement_start( Start ),
    knowledge_bases( KB ),
    amalgamation_statement_end( End ).

knowledge_bases(
  knowledge_bases( [ KB ] ) ) -->
    knowledge_base( KB ).

knowledge_bases(
  knowledge_bases( [ KB | KBs ] ) ) -->
    knowledge_base( KB ),
    [ '&&' ],
    knowledge_bases( knowledge_bases( KBs ) ).

flogic_query(
  flogic_query( X ) ) -->
    flogic_rule_body( X ),
    [ '.' ].

flogic_rule_head(
  flogic_rule_head( X ) ) -->
    flogic_list_of_molecules( X ).

flogic_rule_body(
  flogic_rule_body( B ) ) -->
    flogic_list_of_literals( B ).

flogic_list_of_molecules(
  flogic_list_of_molecules( [ X ] ) ) -->
    flogic_molecule( X ).

flogic_list_of_molecules(
  flogic_list_of_molecules( [ X | Y ] ) ) -->
```

```
    flogic_molecule( X ),
    [ ',' ],
    flogic_list_of_molecules( flogic_list_of_molecules( Y ) ).



flogic_list_of_literals(
  flogic_list_of_literals( [ X ] ) ) -->
    flogic_literal( X ).

flogic_list_of_literals(
  flogic_list_of_literals( [ X | Y ] ) ) -->
    flogic_literal( X ),
    [ ',' ],
    flogic_list_of_literals( flogic_list_of_literals( Y ) ).

flogic_literal(
  flogic_literal( 'not', X ) ) -->
    [ 'not' ],
    flogic_molecule( X ).

flogic_literal(
  flogic_literal( 'not', X ) ) -->
  [ 'not' ],
  [ '(' ],
  flogic_molecule( X ), [ ')' ].

flogic_literal(
  flogic_literal( X ) ) -->
    flogic_molecule( X ).

flogic_molecule(
  flogic_molecule( X ) ) -->
    flogic_fmolecule( X ).
```

```
flogic_molecule(
  flogic_molecule( X ) ) -->
    flogic_pmolecule( X ).

flogic_pmolecule(
  flogic_pmolecule( P ) ) -->
    flogic_predicate( P ).

flogic_pmolecule(
  flogic_pmolecule( P, L ) ) -->
    flogic_predicate( P ),
    [ '(' ],
    flogic_list_of_expressions( L ),
    [ ')' ].

flogic_pmolecule(
  flogic_pmolecule( B ) ) -->
    flogic_build_in_predicate( B ).

flogic_pmolecule(
  flogic_pmolecule( B, L ) ) -->
    flogic_build_in_predicate( B ),
    [ '(' ],
    flogic_list_of_expressions( L ),
    [ ')' ].

flogic_pmolecule(
  flogic_pmolecule( A1, O, A2 ) ) -->
    flogic_arithmetic_expression( A1 ),
    flogic_build_in_infix_predicate( O ),
    flogic_arithmetic_expression( A2 ).

flogic_list_of_expressions(
  flogic_list_of_expressions( [ E ] ) ) -->
```

```
    flogic_expression( E ).

flogic_list_of_expressions(
  flogic_list_of_expressions( [ E | R ] ) ) -->
    flogic_expression( E ),
    [ ',' ],
    flogic_list_of_expressions(
    flogic_list_of_expressions( R ) ).

flogic_expression(
  flogic_expression( P ) ) -->
    flogic_path_expression( P ).

flogic_expression(
  flogic_expression( F ) ) -->
    flogic_fmolecule( F ).
flogic_expression(
  flogic_expression( A ) ) -->
    flogic_aggregrate( A ).

flogic_arithmetic_expression(
  flogic_arithmetic_expression( E ) ) -->
    flogic_expression( E ).

flogic_arithmetic_expression(
  flogic_arithmetic_expression( A1, O, A2 ) ) -->
    flogic_arithmetic_expression( A1 ),
    flogic_build_in_operator( O ),
    flogic_arithmetic_expression( A2 ).

flogic_arithmetic_expression(
  flogic_arithmetic_expression( E ) ) -->
    [ '(' ],
    flogic_arithmetic_expression( E ),
```

```
    [ ')' ].

flogic_aggregrate(
  flogic_aggregrate( Agg, AV, Q ) ) -->
    flogic_id_term( Agg ),
    [ '{' ],
    flogic_variable( AV ),
    [ '|' ],
    flogic_list_of_literals ,
    [ '}' ].

flogic_aggregrate(
  flogic_aggregrate( Agg, AV, GV, Q ) ) -->
    flogic_id_term( Agg ),
    [ '{' ],
    flogic_variable( AV ),
    [ '[' ],
    flogic_list_of_variables( GV ),
    [ ']' ],
    [ '|' ],
    flogic_list_of_literals ,
    [ '}' ].

flogic_fmolecule(
  flogic_fmolecule( P, S ) ) -->
    flogic_path_expression( P ),
    flogic_specification( S ).

flogic_path_expression(
  flogic_path_expression( Id ) ) -->
    flogic_id_term( Id ).

flogic_path_expression(
  flogic_path_expression( E ) ) -->
```

```
    [ '(' ],
    flogic_expression( E ),
    [ ')' ].

flogic_path_expression(
  flogic_path_expression( E, D, M ) ) -->
    flogic_path_expression( E ),
    flogic_dot( D ),
    flogic_method_application( M ).

flogic_path_expression(
  flogic_path_expression( F, D, M ) ) -->
    flogic_fmolecule( F ),
    flogic_dot( D ),
    flogic_method_application( M ).

flogic_specification(
  flogic_specification( I ) ) -->
    flogic_isa_specification( I ).

flogic_specification(
  flogic_specification( I ) ) -->
    flogic_isa_specification( I ),
    [ '[' ],
    [ ']' ].

flogic_specification(
  flogic_specification( I, M ) ) -->
    flogic_isa_specification( I ),
    [ '[' ],
    flogic_list_of_methods( M ),
    [ ']' ].

flogic_specification(
```

```
  flogic_specification( [] ) ) -->
    [ '[' ],
    [ ']' ].

flogic_specification(
  flogic_specification( M ) ) -->
    [ '[' ],
    flogic_list_of_methods( M ),
    [ ']' ].

flogic_isa_specification(
  flogic_isa_specification( S, I ) ) -->
    flogic_isa_symbol( S ),
    flogic_id_term( I ).

flogic_isa_specification(
  flogic_isa_specification( S, E ) ) -->
    flogic_isa_symbol( S ),
    [ '(' ],
    flogic_expression( E ),
    [ ')' ].

flogic_method_application(
  flogic_method_application( I ) ) -->
    flogic_id_term( I ).

flogic_method_application(
  flogic_method_application( I, L ) ) -->
    flogic_id_term( I ),
    [ '(' ],
    flogic_list_of_expressions( L ),
    [ ')' ]. % flora2 doesn't use the '@' sign to separate ↘
    →methods from their arguments since with HiLog extensions ↘
    →it became redundant (floraManual, p. 14)
```

```
flogic_method_application(
  flogic_method_application( E ) ) -->
    [ '(' ],
    flogic_expression( E ),
    [ ')' ].

flogic_method_application(
  flogic_method_application( E, L ) ) -->
    [ '(' ],
    flogic_expression( E ),
    [ ')' ],
    [ '(' ],
    flogic_list_of_expressions( L ),
    [ ')' ].

flogic_list_of_methods(
  flogic_list_of_methods( [ MS ] ) ) -->
    flogic_method_specification( MS ).

flogic_list_of_methods(
  flogic_list_of_methods( [ MS | L ] ) ) -->
    flogic_method_specification( MS ),
    [ ',' ],
    flogic_list_of_methods( flogic_list_of_methods( L ) ).

flogic_method_specification(
  flogic_method_specification( Ma, Mr ) ) -->
    flogic_method_application( Ma ),
    flogic_method_result( Mr ).

flogic_method_specification(
  flogic_method_specification( 'not', Ma, Mr ) ) -->
  [ 'not' ],
```

```
  [ '(' ],
  flogic_method_application( Ma ),
  flogic_method_result( Mr ),
  [ ')' ].

flogic_method_result(
  flogic_method_result( Arr, E ) ) -->
    flogic_method_arrow1( Arr ),
    flogic_expression( E ).

flogic_method_result(
  flogic_method_result( Arr, E ) ) -->
    flogic_method_arrow1( Arr ),
    [ '{' ],
    flogic_list_of_expressions( E ),
    [ '}' ].

flogic_method_result(
  flogic_method_result( Arr, E ) ) -->
  flogic_method_arrow2( Arr ),
  flogic_expression( E ).

flogic_method_result(
  flogic_method_result( Arr, E ) ) -->
    flogic_method_arrow2( Arr ),
    [ '{' ],
    flogic_list_of_expressions( E ),
    [ '}' ].

flogic_id_term(
  flogic_id_term( B ) ) -->
    flogic_basic_id_term( B ).

flogic_id_term(
```

```
  flogic_id_term( F, E ) ) -->
    flogic_functor( F ),
    [ '(' ],
    flogic_list_of_expressions( E ),
    [ ')' ].

flogic_basic_id_term(
  flogic_basic_id_term( F ) ) -->
    flogic_functor( F ).

flogic_basic_id_term(
  flogic_basic_id_term( V ) ) -->
    flogic_variable( V ).

flogic_basic_id_term(
  flogic_basic_id_term( S ) ) -->
    flogic_string( S ).

flogic_basic_id_term(
  flogic_basic_id_term( I ) ) -->
    flogic_integer( I ).

flogic_list_of_variables(
  flogic_list_of_variables( [ X ] ) ) -->
    flogic_variable( X ).

flogic_list_of_variables(
  flogic_list_of_variables( [ X | Y ] ) ) -->
    flogic_variable( X ),
    [ ',' ],
    flogic_list_of_variables( flogic_list_of_variables( Y ) ).

% meta info grammar
```

```
meta_info(
  meta_info( [ T @ P ] ) ) -->
    attribute_value_tag( T ),
    [ '(' ],
    probability_value( P ),
    [ ')' ].

meta_info( meta_info( [ L @ P ] ) ) -->
  hyperlink( L ),
  [ '(' ],
  probability_value( P ),
  [ ')' ].

meta_info(
  meta_info( [ T @ P | R ] ) ) -->
    attribute_value_tag( T ),
    [ '(' ],
    probability_value( P ),
    [ ')' ],
    meta_info( meta_info( R ) ).

meta_info(
  meta_info( [ L @ P | R ] ) ) -->
    hyperlink( L ),
    [ '(' ],
    probability_value( P ),
    [ ')' ],
    meta_info( meta_info( R ) ).

attribute_value_tag(
  attribute_value_tag( A, V ) ) -->
    attribute( A ),
    [ ':' ],
    value( V ).
```

```
attribute_value_tag(
  attribute_value_tag( class, V ) ) -->
    [ class ],
    [ ':' ],
    flogic_id_term( V ).

attribute_value_tag(
  attribute_value_tag( subclass, V ) ) -->
    [ subclass ],
    [ ':' ],
    flogic_id_term( V ).

attribute_value_tag(
  attribute_value_tag( subclass, E ) ) -->
    [ subclass ],
    [ ':' ],
    [ '(' ],
    flogic_expression( E ),
    [ ')' ].

attribute_value_tag(
  attribute_value_tag( rule, H, I, B ) ) -->
    [ rule ],
    [ ':' ],
    flogic_rule_head( H ),
    flogic_implication_symbol( I ),
    flogic_rule_body( B ).

attribute(
  attribute( A ) ) -->
    flogic_id_term( A ).

value(
```

```
  value( V ) ) -->
    flogic_id_term( V ).

internal_url(
  internal_url( U ) ) -->
    flogic_id_term( U ).

external_url(
  external_url( U ) ) -->
    flogic_id_term( U ).

% parsing & auxilliary predicates

pprint( Term ) :-
  term_to_atom( Term, Atom ),
  str_cat( 'python pprint.py ''', Atom, Part ),
  str_cat( Part, '''', Command ),
  shell( Command ).

read_code( List ) :-
  shell_to_list( 'python readcode.py', [ List ], _ ).

parse :-
  read_code( X ),
  autopoiesis_facilitating_semantic_wiki_page( T, X, [] ),
  pprint( T ).

parse_t :-
  read_code( X ),
  trace,
  autopoiesis_facilitating_semantic_wiki_page( T, X, [] ),
  notrace,
  pprint( T ).
```

```
?- parse.
```

Appendix D

Annotated Query Execution Engine Implementation Issues

In order to implement annotated query execution three things had to be considered:

- ISA-expressions
- Rule executions
- Query parsing

ISA expressions are a special case since all paths from some class a to some class b have to be considered when evaluating $a::b$ probability. In order to find all paths the following $\mathcal{F}$LORA-*2* program was implemented

```
?- _optimize(class_expressions).

strict_sub( ?x, ?y ) :-
  ?x :: ?y,
  not( not_strict( ?x, ?y ) ).

not_strict( ?x, ?y ) :-
  ?x :: ?z,
  ?z :: ?y,
  not( ?z = ?y ),
  not( ?z = ?x ).

path( [ p( ?c1, ?c2 ) ], ?c1, ?c2 ) :-
```

```
  strict_sub( ?c1, ?c2 ).

path( [ p( ?c1, ?x ) | ?r ], ?c1, ?c2 ) :-
  strict_sub( ?c1, ?x ),
  path( ?r, ?x, ?c2 ).
```

As defined in chapter 7 rule probabilities are calculated depending on the number of times the rule was executed to yield a given result. Since $\mathcal{F}$LORA-*2* is completely declarative and tabled such a rule counting mechanism is no easy task to implement. Such a mechanism could be implememented inside the $\mathcal{F}$LORA-*2* compiler or inside the XSB prolog engine. For our case we used a little trick to count the number of times a rule was executed for a given result. First a query is normally executed, and then the results are inserted into the appropriate variables that bound to them. All rules have an `increment(rule_id)` statement appended on their body end. Then every result is executed as a query woth the following predicate added to the knowledgebase.

```
increment( ?r ) :-
  rc( ?r, ?c ),
  delete{ rc( ?r, ?c ) },
  ?c1 is ?c + 1,
  insert{ rc( ?r, ?c1 ) }.
```

Any query has to be parsed according to the rules defined in chapter 7 so probabilities can be annotated and calculated for given results. This is done using the BNF grammar for F-Logic and the following predicates:

```
parse( F, R ) :-
  flogic_query( T, F, [] ),
  prs( T, R ).

parse_full( F, R ) :-
  parse( F, R1 ),
  obj_prs( R1, R ).

obj_prs( F, R ) :-
  F = obj( O, F1 ) * ( F2 ),
  obj_prs( obj( O, F1 ), R1 ),
```

```
  obj_prs( F2, R2 ),
  R = R1 * R2.

obj_prs( F, R ) :-
  F = obj( O, F1 ) + ( F2 ),
  obj_prs( obj( O, F1 ), R1 ),
  obj_prs( F2, R2 ),
  R = R1 + R2.

obj_prs( obj( O, F ), R ) :-
  F = min( p( att, V1 ), p( val, V2 ) ) * ( F1 ),
  obj_prs( obj( O, F1 ), R1 ),
  R = min( p( O, att, V1 ), p( O, val, V2 ) ) * R1.

obj_prs( obj( O, F ), R ) :-
  F = min( p( att, V1 ), p( val, V2 ) ),
  R = min( p( O, att, V1 ), p( O, val, V2 ) ).

obj_prs( obj( O, F ), R ) :-
  F = p( O, I, C ) * ( F1 ),
  obj_prs( obj( O, F1 ), R1 ),
  R = p( O, I, C ) * R1.

obj_prs( obj( O, F ), R ) :-
  F = p( O, I, C ),
  R = p( O, I, C ).

obj_prs( obj( O, F ), R ) :-
  F = min( p( att, V1 ), p( val, V2 ) ) + ( F1 ),
  obj_prs( obj( O, F1 ), R1 ),
  R = min( p( O, att, V1 ), p( O, val, V2 ) ) + R1.

obj_prs( obj( O, F ), R ) :-
  F = p( O, I, C ) + ( F1 ),
```

```
  obj_prs( obj( O, F1 ), R1 ),
  R = p( O, I, C ) + R1.

prs( flogic_query( Q ), P ) :-
  prs( Q, P ).

prs( flogic_rule_body( B ), P ) :-
  prs( B, P ).

prs( flogic_list_of_literals( L ), P ) :-
  prs( L, P ).

prs( [ H, ',' | R ], P ) :-
  prs( H, P1 ),
  prs( R, P2 ),
  P = P1 * P2.

prs( [ H, ';' | R ], P ) :-
  prs( H, P1 ),
  prs( R, P2 ),
  P = P1 + P2.

prs( [ L ], P ) :-
  prs( L, P ).



prs(
  flogic_literal(
    flogic_molecule(
      flogic_fmolecule(
        flogic_path_expression(
          flogic_id_term(
            flogic_basic_id_term(
              O
```

```
            )
          )
        ),
        flogic_specification(
          flogic_isa_specification(
            flogic_isa_symbol( S ),
            flogic_id_term(
              flogic_basic_id_term(
                C
              )
            )
          )
        )
      )
    )
  ),
  P ) :-
    prs( O, OP ),
    prs( C, OC ),
    P = obj( OP, p( OP, S, OC ) ).

prs(
  flogic_literal(
    flogic_molecule(
      flogic_fmolecule(
        flogic_path_expression(
          flogic_id_term(
            flogic_basic_id_term(
              O
            )
          )
        ),
        flogic_specification(
          flogic_isa_specification(
```

```
            flogic_isa_symbol( S ),
            flogic_id_term(
              flogic_basic_id_term(
                C
              )
            )
          ),
          ML
        )
      )
    )
  ),
  P ) :-
    prs( ML, P1 ),
    prs( O, OP ),
    prs( C, OC ),
    P = obj( OP, p( OP, S, OC ) * P1 ).

prs(
  flogic_literal(
    flogic_molecule(
      flogic_fmolecule(
        flogic_path_expression(
          flogic_id_term(
            flogic_basic_id_term(
              O
            )
          )
        ),
        flogic_specification(
          ML
        )
      )
    )
```

```
  ), P ) :-
    prs( ML, PM ),
    prs( O, PO ),
    P = obj( PO, PM ).

prs( flogic_string( flogic_variable( V ) ), V ).
prs( flogic_string( S ), S ) :-
  not( flogic_variable( _ ) = S ).

prs( flogic_variable( V ), V ).

prs( flogic_functor( flogic_variable( V ) ), V ).
prs( flogic_functor( F ), F ) :-
  not( flogic_variable( _ ) = F ).

prs(
  flogic_method_specification(
    flogic_method_application(
      flogic_id_term(
        flogic_basic_id_term(
          A
        )
      )
    ),
    flogic_method_result(
      Arr,
      flogic_expression(
        flogic_path_expression(
          flogic_id_term(
            flogic_basic_id_term(
              V
            )
          )
        )
      )
```

```
      )
    )
  ), P ) :-
    prs( A, PA ),
    prs( V, PV ),
    P = min( p( att, PA ), p( val, PV ) ).

prs( flogic_list_of_methods( [ H ] ), P ) :-
  prs( H, P ).

prs( flogic_list_of_methods( [ H | T ] ), P ) :-
  prs( H, P1 ),
  prs( flogic_list_of_methods( T ), P2 ),
  P = P1 * P2.
```

Appendix E

┬aOPīs Source Code

The architecture of ┬aOPīs is shown on figure E.1. The system basically consists of a graphical user interface (written in PHP and partially AJAX), a database interface (written mostly in PL/PGSQL - the procedural language of PostgreSQL and PL/PythonU - the Python procedural language for PostgreSQL), a database (written in PostgreSQL), an integration layer (written in Python), a knowledge base (written and generated into $\mathcal{F}$LORA-*2*) as well as other technologies like web services, mailing lists, podcasting services, content feeds etc. which were interfaced with Python.

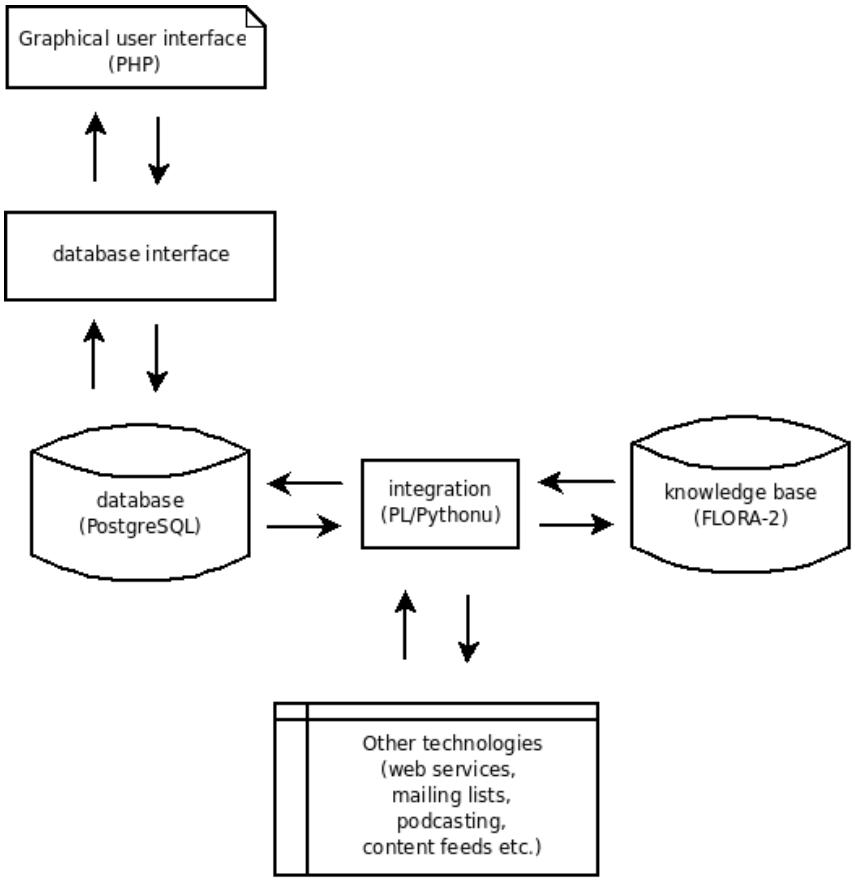

Figure E.1: ┬aOPīs system's architecture

The graphical user interface mainly acts as a presentation layer for the system running in the background. Most functionality is implemented in the database itself using stored procedures. The most significant parts of the implementation include the **niKlas** language parser that translates **niKlas** syntax into HTML. The parser also takes care of possible queries that have to be executed before the rest of the syntax is interpreted. On the other hand the integration layer in also of importance. It was written in Python and allows a direct interface between PostgreSQL and the $\mathcal{F}$LORA-*2* reasoning engine. Queries written in **niKlas** syntax are executed towards a dynamically generated knowledge base for a given semantic wiki system. This generator is the third most significant part of the implementation (but it has to be mentioned that an OWL generator was implemented as well, but can only be used to export the not annotated nor amalgamated OWL ontology). Part of the generator and query executor had to be written in $\mathcal{F}$LORA-*2* due to special situations.

The ΤAOPĪS source code is organized as shown in the following directory tree. Basically there are two important parts of the source code: (1) the database (files in folder sql) and (2) the PHP application (root directory). The database files contains the most important parts of ΤAOPĪS , whilst the PHP application represents the presentation layer. Due to the separation of functionality and presentation layers new graphical user interfaces can be easily build using the functionality layer as an API. To demonstrate this a Facebook application was build that allows users to connect to ΤAOPĪS from their Facebook account. ΤAOPĪS is open source and available at `http://autopoiesis.foi.hr`.

```
|-- CVS
|   |-- Entries
|   |-- Repository
|   `-- Root
|-- CVSROOT
|   |-- CVS
|   |   |-- Entries
|   |   |-- Repository
|   |   `-- Root
|   |-- checkoutlist
|   |-- checkoutlist,v
|   |-- commitinfo
|   |-- commitinfo,v
```

```
|   |-- config
|   |-- config,v
|   |-- cvswrappers
|   |-- cvswrappers,v
|   |-- history
|   |-- loginfo
|   |-- loginfo,v
|   |-- modules
|   |-- modules,v
|   |-- notify
|   |-- notify,v
|   |-- postadmin
|   |-- postadmin,v
|   |-- postproxy
|   |-- postproxy,v
|   |-- posttag
|   |-- posttag,v
|   |-- postwatch
|   |-- postwatch,v
|   |-- preproxy
|   |-- preproxy,v
|   |-- rcsinfo
|   |-- rcsinfo,v
|   |-- taginfo
|   |-- taginfo,v
|   |-- val-tags
|   |-- verifymsg
|   `-- verifymsg,v
|-- blog.php
|-- clean.sh
|-- config.php
|-- edit-wiki.php
|-- emoticons.php
|-- favicon2.ico
```

```
|-- filter.php
|-- flora2export.php
|-- footer.php
|-- forum.php
|-- header.php
|-- history.php
|-- i18n.php
|-- image_wrapper.php
|-- images
|   |-- CVS
|   |   |-- Entries
|   |   |-- Repository
|   |   `-- Root
|   |-- arrowdown.gif
|   |-- arrowleft.gif
|   |-- avatars
|   |   |-- CVS
|   |   |   |-- Entries
|   |   |   |-- Repository
|   |   |   `-- Root
|   |   `-- anonimno1.jpg
|   |-- bubles2.png
|   |-- ccLicense.png
|   |-- colourful.jpg
|   |-- icons
|   |   |-- CVS
|   |   |   |-- Entries
|   |   |   |-- Repository
|   |   |   `-- Root
|   |   |-- flora2.png
|   |   |-- forum.png
|   |   |-- join.png
|   |   |-- leave.png
|   |   |-- long time no see.gif
```

```
|   |   |-- minus.gif
|   |   |-- new.png
|   |   |-- offline.gif
|   |   |-- online.gif
|   |   |-- owl.png
|   |   |-- plus.gif
|   |   |-- stop.gif
|   |   `-- wiki.png
|   |-- icontexto-webdev-emoticon-sad-032x032.png
|   |-- icontexto-webdev-emoticon-smile-032x032.png
|   |-- icontexto-webdev-info-032x032.png
|   |-- icontexto-webdev-ok-032x032.png
|   |-- light-balance-small.jpg
|   |-- logo.gif
|   |-- logo.png
|   |-- orangefilter.gif
|   |-- redfilter.gif
|   |-- smiles
|   |   |-- 156.gif
|   |   |-- 1_4_126.gif
|   |   |-- 2thumbs.gif
|   |   |-- 3zzz.gif
|   |   |-- 45.gif
|   |   |-- CVS
|   |   |   |-- Entries
|   |   |   |-- Repository
|   |   |   `-- Root
|   |   |-- Hail.gif
|   |   |-- Party_fest25.gif
|   |   |-- Peglaona.gif
|   |   |-- Pop.gif
|   |   |-- Prd.gif
|   |   |-- Sex.gif
|   |   |-- Slin.gif
```

```
|   |   |-- Thumbs.db
|   |   |-- afro.gif
|   |   |-- amen.gif
|   |   |-- angel.gif
|   |   |-- angel10.gif
|   |   |-- angeleye.gif
|   |   |-- angry.gif
|   |   |-- azdaja.gif
|   |   |-- bananallama.gif
|   |   |-- bara.gif
|   |   |-- bau.gif
|   |   |-- beerchug.gif
|   |   |-- bicedobro.gif
|   |   |-- biggrin.gif
|   |   |-- blabla.gif
|   |   |-- blush.gif
|   |   |-- bonk.gif
|   |   |-- bootyshake.gif
|   |   |-- braca.gif
|   |   |-- buum.gif
|   |   |-- ceka.gif
|   |   |-- cheer.gif
|   |   |-- cice.gif
|   |   |-- clint.gif
|   |   |-- confused.gif
|   |   |-- cycle.gif
|   |   |-- d020.gif
|   |   |-- d062.gif
|   |   |-- daz.gif
|   |   |-- dinamo.gif
|   |   |-- dog.gif
|   |   |-- downtown.gif
|   |   |-- driver.gif
|   |   |-- duckie.gif
```

```
|   |   |-- dvoboj.gif
|   |   |-- e015.gif
|   |   |-- eek.gif
|   |   |-- evil1.gif
|   |   |-- evil3.gif
|   |   |-- gitara.gif
|   |   |-- headbang.gif
|   |   |-- hebemu.gif
|   |   |-- hekla.gif
|   |   |-- horor.gif
|   |   |-- icon_anal.gif
|   |   |-- icon_pall.gif
|   |   |-- icon_toilet.gif
|   |   |-- icon_weed.gif
|   |   |-- jerk.gif
|   |   |-- jumping.gif
|   |   |-- kada.gif
|   |   |-- kava.gif
|   |   |-- kiss.gif
|   |   |-- klap.gif
|   |   |-- klopa.gif
|   |   |-- kukuc.gif
|   |   |-- kusch.gif
|   |   |-- lista.txt
|   |   |-- lol.gif
|   |   |-- love2.gif
|   |   |-- lurk.gif
|   |   |-- mad.gif
|   |   |-- mama.gif
|   |   |-- mar.gif
|   |   |-- minitiere067.gif
|   |   |-- mirko.gif
|   |   |-- misli.gif
|   |   |-- moli.gif
```

```
|   |   |-- ne_zna.gif
|   |   |-- ninja.gif
|   |   |-- nono.gif
|   |   |-- off.gif
|   |   |-- old.gif
|   |   |-- osama.gif
|   |   |-- p020.gif
|   |   |-- pila.gif
|   |   |-- pljesko.gif
|   |   |-- popc1.gif
|   |   |-- puke.gif
|   |   |-- redface.gif
|   |   |-- rigo014.gif
|   |   |-- rock.gif
|   |   |-- rodendan.gif
|   |   |-- rofl.gif
|   |   |-- romeo.gif
|   |   |-- shakecan.gif
|   |   |-- shhh.gif
|   |   |-- sm_bigeek.gif
|   |   |-- sm_biggrin.gif
|   |   |-- sm_confused.gif
|   |   |-- sm_cool.gif
|   |   |-- sm_cry.gif
|   |   |-- sm_dead.gif
|   |   |-- sm_mad.gif
|   |   |-- sm_razz.gif
|   |   |-- sm_rolleyes.gif
|   |   |-- sm_sigh.gif
|   |   |-- sm_sleep.gif
|   |   |-- sm_smile.gif
|   |   |-- sm_upset.gif
|   |   |-- sm_wink.gif
|   |   |-- smash.gif
```

```
|   |   |-- smilie_flagge19.gif
|   |   |-- sok.gif
|   |   |-- stirka.gif
|   |   |-- supak.gif
|   |   |-- thumbs.gif
|   |   |-- thumbsdown.gif
|   |   |-- traca.gif
|   |   |-- trio.gif
|   |   |-- troll.gif
|   |   |-- tuctuc.gif
|   |   |-- tulum.gif
|   |   |-- vidiga.gif
|   |   |-- whacky091.gif
|   |   |-- whatever.gif
|   |   |-- world_domination.gif
|   |   |-- zivili.gif
|   |   |-- zjev.gif
|   |   |-- zubeki.gif
|   |   |-- zubo.gif
|   |   |-- zvrko.gif
|   |   `-- zzz.gif
|   |-- spider-left.jpg
|   |-- spider-right.jpg
|   `-- yellowfilter.gif
|-- index.php
|-- install.php
|-- install_flora.php
|-- install_xsb.php
|-- join.php
|-- login.php
|-- lost_pass.php
|-- menu.php
|-- mysettings.php
|-- owlexport.php
```

```
|-- post-message.php
|-- post.php
|-- project.php
|-- register-proorg.php
|-- register-user.php
|-- register.php
|-- search.php
|-- security_image.php
|-- sql
|   |-- CVS
|   |   |-- Entries
|   |   |-- Repository
|   |   `-- Root
|   |-- connection.php
|   |-- taopis\_create.sql
|   `-- taopis_drop.sql
|-- style.php
|-- style_settings.php
|-- styles
|   |-- CVS
|   |   |-- Entries
|   |   |-- Repository
|   |   `-- Root
|   |-- ben
|   |   |-- CVS
|   |   |   |-- Entries
|   |   |   |-- Repository
|   |   |   `-- Root
|   |   |-- images
|   |   |   |-- CVS
|   |   |   |   |-- Entries
|   |   |   |   |-- Repository
|   |   |   |   `-- Root
|   |   |   |-- greyfilter.gif
```

```
|   |   |   `-- statue.jpg
|   |   `-- top.css
|   |-- taopis_aqua
|   |   |-- CVS
|   |   |   |-- Entries
|   |   |   |-- Repository
|   |   |   `-- Root
|   |   |-- images
|   |   |   |-- CVS
|   |   |   |   |-- Entries
|   |   |   |   |-- Repository
|   |   |   |   `-- Root
|   |   |   |-- bluefilter.gif
|   |   |   `-- bubles.png
|   |   `-- top.css
|   `-- taopis_red_sea
|       |-- CVS
|       |   |-- Entries
|       |   |-- Repository
|       |   `-- Root
|       |-- images
|       |   |-- CVS
|       |   |   |-- Entries
|       |   |   |-- Repository
|       |   |   `-- Root
|       |   |-- back_redsea.png
|       |   `-- redfilter.gif
|       `-- top.css
|-- suggest.php
|-- tag.php
|-- top_editor.js
|-- upload_avatar.php
|-- user.php
|-- userlist.php
```

```
'-- wiki.php
```

The rest of the source code has been ommitted here, but available on the CD-ROM which is an integral part of this appendix.

Sažetak

Wiki sustavi, progresivna tehnologija kojoj u njezinim počecima neki poznati stručnjaci nisu predvidjeli svijetlu budućnost, danas su u širokoj upotrebi. Sustavi koji svakom pridošlici omogućavaju da na njima ostavi traga, razvijaju se autopoietično u sve impresivnije i impresivnije repozitorije znanja. Možda najpoznatiji primjer takvog sustava, Wikipedia, otvorena enciklopedija Interneta u vrijeme pisanja ovog teksta u svojoj engleskoj inačici broji preko 2.2 milijuna članaka koje su ljudi širom svijeta postavili na sustav, a postoje inačice za gotovo sve svjetske jezike.

Ipak, čini se da su wiki sustavi došli do svoje granice rasta. Sve je češće i češće slučaj da se na različitim sustavima pokušavaju definirati pravila ponašanja, pravila organiziranja znanja, pravila dodavanja metapodataka primarno u svrhu jednostavnijega pretraživanja i izvođenja zaključaka iz ovih ogromnih repozitorija (uglavnom) tekstualnih podataka.

Napori poput semantičkih wiki sustava, koji u tradicionalne (obične) wiki sustave pokušavaju dodati semantičku komponentu čini se u potpunosti zanemaruju jedan od osnovnih razloga nevjerojatnog uspjeha ove vrste sustava. Wiki sustavi su jednostavni za korištenje i stoga ih koristi širok spektar ljudi. Korisnici imaju vrlo različita shvaćanja tehnologije koja variraju od vrhunskih stručnjaka za informacijsku tehnologiju do laika. Dakako, da distribucija korisnika naginje onima manje vičnim informacijskim tehnologijama. Upravo zbog toga uvođenje naprednih koncepata poput semantičkih tehnologija uvelike ograničava primjenjivost takvih sustava jer od običnih korisnika traži relativno dobro poznavanje takvih tehnologija.

Kao što je prethodno napomenuto, wiki sustavi razvijaju se autopoietično, za razliku od tradicionalnih alopoietičnih (tehničkih) informacijskih tehnologija. Autopoiesis očitava se upravo u činjenici da korisnici svojim sudjelovanjem na sustavu stvaraju taj sustav, šire ga, unaprijeđuju ga novim i novima sadržajima, pravilima i definicijama. Postavlja se pitanje je li moguće koncept semantičkog weba "ugraditi" u wiki sustave, a da se pri tome zadrži njihova početna jednostavnost?

Jedna druga vrsta suvremenih Web 2.0 sustava na koje želimo ovdje ukazati su sustavi za društveno označavanje (engl. social tagging). Oni su danas sve češće u upotrebi, a koriste upravo organizaciji znanja pojedinog korisnika (engl. personal information management – PIM; personal knowledge management – PKM). Takvi sustavi svojim korisnicima omogućavaju da postavljaju oznake (engl. tag) na bilo koji sadržaj na koji nailaze na webu. Impresivni su takvi sustavi iz perspektive pretraživanja. Naime dok poznate tražilice pretražuju web naprednim algoritmima, sustavi za društveno označavanje koriste jednostavno oznake koje su postavili korisnici. Sustavi za društveno označavanje često pronalaze relevantnije podatke od naprednih tražilica jer dolazi to tzv. Delfi efekta prema kojem je prosječno mišljenje nekog podskupa ljudi bolji prediktor od mišljenja jedne nasumice odabrane osobe.

Još jedno pitanje koje ovdje valja postaviti jest pitanje suvremenih organizacija i posebice njihovih informacijskih sustava. Suvremene organizacije danas su otvorene, adaptibilne, heterarhijske i virtualne. Je li moguće iskoristiti suvremene informacijske tehnologije kako bi se podržale potrebe suvremenih organizacija za adaptibilnošću, otvorenošću, heterarhiji i virtualnošću? U ovom ćemo radu pokušalo se, pa makar djelomično, odgovoriti i na to pitanje, na koje suvremeni (rigidni, alopoietski postavljeni) informacijski sustavi ne daju odgovor.

Web usluge (engl. web services) danas su način na koji je putem mreže moguće koristiti usluge raznih organizacija. Semantičkim opisom takvih usluga pokušava se omogućiti automatizirano računalno korištenje takvih usluga. Je li integracijom ove tehnologije i gore navedenih moguće podržati potrebe suvremenih organizacija?

U ovom radu krenulo se pristupom objektno-orijentiranog semantičkog modeliranja te su tako dobivena saznanja stavljena u autopoietični kontekst. Osnovni ciljevi bili su: (1) da sustav autopoietično generira formalizirano znanje nad kojim se može računalno rezonirati, (2) od prosječnog korisnika očekuje se nikakvo ili minimalno poznavanje semantičkih tehnologija, (3) sustav se treba autopoietično razvijati kako na području vlastitog sadržaja tako i na području vlastite funkcionalnosti (za razliku od tradicionalnih wiki sustava koji se razvijaju u pravilu isključivo na području sadržaja).

Pretpostavljeno je da je svijet kojeg korisnici na sustavu opisuju jedan skup objekata koji su u međusobnim relacijama i raznim interakcijama. Može se reći da je "osnovna jedinica" sustava objekt. Svaki objekt potencijalno ima svoje relacije s drugim objektima i niz metoda kojima reagira na podražaje (poruke) od drugih objekata. Relacije se ponekad u kontekstu objektno-orijentiranog pristupa nazivaju i atributima radi jednos-

tavnosti implementacije iako je riječ o relaciji sadržavanja. U radu je također primjenjen takav pristup radi jednostavnosti te se atributima smatraju objekti koji su jednostavni znakovni nizovi dok će svi ostali objekti biti vezani relacijama. Podskupove skupa svih objekata nazivamo ekstenzijom nekog koncepta ukoliko postoji jasna intenzija (definicija, pravilo) koncepta po kojem jednoznačno možemo svaki objekt klasificirati bilo kao člana ekstenzije koncepta, bilo kao člana komplementarnog skupa ekstenziji. Koncepti se još nazivaju tipovima ili klasama te ćemo ta tri naziva u daljnjem razmatranju smatrati sinonimima. Možemo reći da se svaki koncept sastoji od svoje intenzije (definicije), ekstenzije (skupa svih objekata na koje je koncept primjenjiv) i svog simbola (oznakom kojom označavamo taj specifični koncept).

Ako se promotri wiki sustav iz perspektive semantičkog modeliranja može se reći da postoje tri osnovna koncepta na kojima je moguće temeljiti razmatranje, a to su: (1) stranica, (2) osoba i (3) wiki sustav (koji uključuje sam sustav, sve njegove članke odnosno stranice i korisnike). Uvedena je sljedeća pretpostavku: neka je svaki objekt tipa stranica generički objekt (u kontekstu modeliranja znanja često označen simbolom Thing). Neka sada svaki korisnik sustava može označavati stranice na sustavu postavljajući oznake u obliku uređenih parova atribut:vrijednost. Na taj način korisnici specijaliziraju svaki generički objekt u neku (novu) klasu objekata. Također, neka atribut i vrijednost mogu biti i neka od ključnih riječi poput class, inherits, relation, rule i sl. poznatih iz objektno-orijentiranih programskih jezika. Na taj način korisnik potencijalno može još uže specijalizirati svoje mišljenje (znanje) o stranici (objektu). Na taj je način podržano dodavanje atributa pojedinom objektu kao i njegovo određivanje koje se tiče klase u koju pripada.

Primjerice ako neki korisnik na neku stranicu postavi oznaku class:avion to znači da taj korisnik tu stranicu više ne smatra člankom već opisnikom objekta tipa "avion". Također, svaki korisnik može i potvrditi postojeću oznaku čime se vjerodostojnost oznake povećava.

Wiki sustavi po svojoj standardnoj sintaksi omogućavaju povezivanje svake stranice s drugim stranicama kako na samom sustavu tako i izvan njega putem hiperveza. Možemo reći da je ovdje riječ o relacijama s drugim objektima. Pretpostavlja se dakle da svaki korisnik može na svaku stranicu dodavati hiperveze na druge stranice u obliku relacija : naziv_objekta pri čemu može biti riječ o stranicama na samom sustavu (koje su onda tipa stranica ili nekog specijaliziranijeg tipa) ili eksternim stranicama (pri čemu uvodimo tip external resource). Neka, također svaki korisnik može potvrditi vezu kao i

kod atributa odnosno označavanja čime se vjerodostojnost relacije povećava.

Pretpostavlja se nadalje da svaki korisnik može svakoj stranici nadodati web uslugu ili neku drugu vrstu skriptne ekstenzije koja proširuje funkcionalnost stranice te takve dodatke nazivamo metodama. Neka je svaka takva metoda semantički opisana pomoću standardnog obrasca i svog opisnika (primjerice engl. WSDL - Web Service Definition Language). Svaka se metoda, kao i u prethodna dva slučaja može potvrditi od drugih korisnika čime se povećava njezina vjerodostojnost.

Zaključno možemo reći da smo ovako koncipiranim sustavom u stanju podržati dinamično kreiranje klasa, objekata, njihovih atributa, metoda i relacija. Konkretno, moglo bi se reći da je riječ o dinamički kreiranoj ontologiji, dakle formalizaciji određene aplikacijske domene.

Pozabavimo se sada vjerodostojnošću (istinitošću) informacija koje korisnici pospremaju u sustav. Kako bi to učinili potrebno je prethodno opisati koncept organizacije ribarske mreže kao i mogućnosti podržavanja takvog koncepta informacijskom tehnologijom. Riječ je o konceptu koji pokušava iskoristiti najbolje od dvaju poznatih koncepata iz organizacijske teorije, hijerarhije i heterarhije odnosno mrežne strukture. Ako promatramo ribarsku mrežu na obali ona se čini potpuno heterarhijskom, svi su čvorovi istovjetni i na jednakoj razini. No primimo li jedan čvor i uzdignemo ga dinamički oko njega nastaje hijerarhija pri čemu je odabrani čvor na vrhu. Primimo li drugi nastaje druga itd. Na taj način dinamički možemo stvarati nove i uništavati stare hijerarhije.

Postavlja se pitanje kako podržati takav koncept informacijskom tehnologijom. Pretpostavimo da imamo sustav na kojem se vodi niz projekata neke organizacije (ili općenito nekog socijalnog sustava) pri čemu svaki projekt ima svoj vlastiti autopoietični semantički wiki sustav. Moglo bi se reći da svaki projekt definira svojevrsnu aplikacijsku domenu. Sama funkcionalnost tog sustava u ovom trenutku nam nije bitna, nego nam je bitan mehanizam kojim ćemo pronaći najadakvatniju osobu za pojedini projekt koja se svojim znanjima i sposobnostima ističe te time postaje vođa.

Kako bi objasnili taj mehanizam potrebno je objasniti PageRank algoritam kojeg koristi poznata tražilica Google pri rangiranju stranica koje se pretražuju. Algoritam analizira web stranice brojeći ulazne i izlazne veze svake stranice. Svaka veza koju neka stranica ima prema nekoj drugoj stranici smatra se "glasom potpore" te stranice za stranicu na koju pokazuje. Sve se stranice inicijalno postavljaju na određeni rank koji je jednak 1/N pri čemu je N broj stranica koje se analiziraju. Ako neka stranica kojim slučajem pokazuje na više stranica tada se njezin glas dijeli tako da svaka stranica dobiva

n-ti dio njezinoga glasa (ako je n broj stranica na koje stranica pokazuje). Sada se do određene preciznosti analizira tako postavljena mreža stranica zbrajanjem ulaznih veza svake stranice i prosljeđivanjem novih vrijednosti na druge stranice. Na taj način dobije se rank za svaku stranicu, tă što je on veći to ta stranica ima veću "potporu" drugih stranica. Taj se rank može nazvati i vjerojatnošću kojom će neki korisnik nasumice obilazeći stranice u zadanoj mreži odabrati upravo zadanu stranicu. Page rank mreža ustvari predstavlja Markovljev lanac u kojem su stranice stanja, a tranzicije (koje su sve jednako vjerojatne) veze između tih stranica.

Postavimo sad taj algoritam u kontekst jedne socijalne mreže, dakle mreže ljudi, odnosno konkretno mreže članova nekog projekta ili aplikacijske domene. Dajemo svim članovima projekta mogućnost da glasuju za druge korisnike za koje smatraju da su najadekvatniji za vođu projekta. Ako sada glasove promatramo kao veze na druge stranice tada primjenom PageRank algoritma dinamički dobivamo hijerarhiju članova na određenom projektu.

Uvišestručavanjem ovog algoritma (stoga i više različitih projekata) u stvari podržavamo koncept ribarske mreže. PageRank algoritam ima još jednu zanimljivu odliku koju ćemo primijeniti u daljnjem izlaganju. Naime, zbroj svih rankova svih čvorova u mreži je jednak 1 što je dakako korisno u kontekstu teorije vjerojatnosti.

Vratimo se sada na istinitost pojedinih informacija u autopoietičnom semantičkom wikiju. Pretpostavimo da je svaka oznaka koju neki korisnik postavi ponderirana njegovim rankom. Ovakav je ponder opravdan jer rank na neki način iskazuje uvjerenje drugih članova da će dotični član učiniti pravu stvar, što lako možemo povezati s definicijom znanja: "Znanje je istinito vjerovanje". Oznake se akumuliraju, dakle svaka se oznaka može postaviti i više puta od različitih članova. Zbrajanjem pondera dobivamo vjerojatnost koja iskazuje uvjerenje članova projekta da je informacija točna.

Sada kad imamo brojčani iznos vjerojatnosti možemo i logički formalizirati znanje u tako dinamički kreiranoj ontologiji. Jasno je da ontologiju možemo opisati nekim od jezika za ontologije (npr. F-Logika – engl. frame logic, f-logic, Deskripcijska Logika – engl. description logic) pri čemu se na ovom mjestu odlučujemo za logiku temeljenu na okvirima odnosno F-Logiku. Potrebno je za odabrani jezik izvesti shemu anotacije vjerojatnosti što će biti učinjeno u radu.

No, što je u slučaju ako zaključke želimo izvoditi iz više različitih ontologija, što je opravdano pitanje. Naime, ako kao što smo pretpostavili imamo niz projekata na kojima se dinamički stvaraju ontologije tada je visoka vjerojatnost da će nam ponekad

biti potrebno znanje iz više različitih područja. Primjerice, recimo da se jedan projekt bavi vinima, drugi receptima i treći gastronomskom ponudom nekog područja. Vrlo jednostavno može se dogoditi da netko postavi upit "U kojem restoranu mogu popiti vino koje najbolje ide uz određenu vrstu ribe, a da pri tome ne platim više od 200 kn?" Agent koji pokušava odgovoriti na to pitanje mora prvo pronaći nazive recepata koji sadrže tu određenu vrstu ribe, zatim mora pronaći vina koja idu uz navedene recepte i na kraju pretražiti restorane koji neku od kombinacija imaju u svojoj gastronomskoj ponudi uz odgovarajuću cijenu.

Za spajanje različitih izvora znanja postoji princip amalgamacije izvora znanja. U radu bi se taj koncept pokušao primijeniti na ovako načinjen sustav ponderiranjem vjerojatnosti pojedinih izraza s izvedenicom broja članova na pojedinom projektu.

U ovom radu izvedena je formalizacija koja povezuje wiki sustave, semantički web, mrežne usluge, društveno označavavanje i socijalne mreže. Uz to opisan je i jedan jezik iz koji je nazvan Niklas (prema poznatom Niklasu Luhmanu koji je prvi uveo pojam autopoiesisa u društvene znanosti). Konkretno, istražene su mogućnosti primjene dobro formaliziranih koncepata iz semantičkog weba i semantičkih mrežnih usluga u autopoietičnom kontekstu wiki sustava uz pripomoć društvenog označavanja i socijalnih mreža kao što je prethodno opisano.

Prvo su objašnjeni i formalizirani jezici za (obične) wiki sustave. U tom kontekstu bilo je potrebno opisati niz formalizacija sintakse koje se u takvim jezicima koriste poput hiperveza, slika i raznih drugih dodataka, poglavlja, formatiranja teksta, tablica, varijabli i predložaka.

Nakon formalizacije jezika za wiki sustave opisani su jezici za semantičke wiki sustave koji su određeno proširenje prethodno opisanih jezika. U tu svrhu bilo je potrebno opisati i formalizirati sintaksu i konkretno koncepte poput atributa i relacija, konverzija, semantičkih predložaka i mrežnih usluga. Pri tome se koristila logika temeljena na okvirima (F-logika) kao jezik za formalizaciju.

Na kraju su semantički wiki sustavi stavljeni u autopoietični kontekst. Bilo je potrebno izvesti anotaciju vjerojatnosti na prethodno definirane jezike obzirom na socijalnu mrežu kao i amalgamacijsku shemu kako bi se omogućila integracija izvora znanja.

Na kraju je prikazan niz primjera mogućih autopoietičnih aplikacija kako bi se poduprla teza o korisnosti prethodno izvedene formalizacije. Najvažniji primjer jest konstrukcija autopoietičnog sustava za sigurnost osobnih računala. Osnovna ideja je izgraditi autopoietični semantički wiki sustav o osnovnim prijetnjama osobnim računalima

(poznatih pogrešaka) te ih povezati s potrebnim zakrpama. Na taj način bi se omogućila svojevrsna baza znanja o prijetnjama i rješenjima te time i razvoj aplikacija za korištenje takve baze. Uz navedeni prikazani su i sustav za autopoietičnu znanstvenu konferenciju ili časopis te autopoietični sustav za upravljanje znanjem u organizaciji.

Glavni cilj ovog istraživanja bio je na jednom mjestu ukazati na teoriju i primjenu autopoiesisa u informacijskim znanostima s posebnim naglaskom na korisnost drukčijeg pristupa semantičkim wiki sustavima. Htjelo se pokazati da su wiki sustavi u načelu autopoietični zbog svoje jednostavnosti korištenja od strane društvenog sustava koji ih okružuje, a ta se jednostavnost gubi uvođenjem kompleksnih semantičkih tehnologija. Svojevrsnim skrivanjem semantičkih tehnologija u pozadinu sustava te uvođenjem drugih društvenih tehnologija poput društvenog označavanja i društvenih mreža pokušao se dokinuti taj jaz između semantičkih tehnologija i wiki sustava.

Hipoteze ovog istraživanja, koje su u radu i potvrđene, bile su sljedeće:

HIPOTEZA 1. Na temelju formalizacije jezika za wiki sustave i jezika za semantičke wiki sustave te uvođenja koncepta organizacije ribarske mreže moguće je izvesti anotacijsku shemu vjerojatnosti u takve jezike po uzoru na druge logičke programske jezike. Anotacijska shema izvedena je uz pomoć sintaske logike temeljene na okvirima.

HIPOTEZA 2. Na temelju sheme za anotaciju vjerojatnosti u jezike za semantičke wiki sustave te na temelju uvođenja analize društvenih mreža u wiki sustave i semantičke wiki sustave moguće je izvesti amalgamacijsku shemu za takve jezike po uzoru na druge logičke programske jezike. Amalgamacijska shema izvedena je uz pomoć integracije društvenih mreža.

HIPOTEZA 3. Na temelju sheme za anotaciju vjerojatnosti i sheme za amalgamaciju moguće je izvesti novu vrstu jezika za autopoietične semantičke wiki sustave. Pri tome je formalizirana sintaksa i semantika novog jezika koji je svojevrsna kombinacija anotirane i amalgamirane logike temeljene na okvirima i prikaznih jezika za wiki sustave.

Iz navedenog jasno je da se znanstveni doprinos ovog rada sastoji od eksplicitne formalizacije (običnih) wiki sustava, formalizacije jezika za semantičke wiki sustave, uvođenja semantičkih mrežnih usluga u semantičke wiki sustave te uvođenja objektno orijentiranog pristupa u formalizaciju jezika za semantičke wiki sustave. Osnovni

doprinos ogledava se u razvoju novog jezika za autopoietične semantičke wiki sustave uvođenjem koncepata dobivenih iz društvenog označavanja i analizom socijalnih mreža odnosno konkretno uvođenjem anotacije tako dobivenih vjerojatnosti i odgovarajuće amalgamacijske sheme te na kraju primjene koncepta autopoiesisa u kontekstu informacijskih sustava. Uz navedeno opisana je jedna implementacija takvog jezika utjelovljena u jeziku niKlas .

Iz društvene perspektive otvara se čitav niz mogućnosti za razvoj novih vrsta aplikacija temeljenih na podržavanju autopoiesisa društvenog sustava. Naime, krajnji proizvod tj. aplikacija ᴛAOPīs koja podržava autopoietične semantičke wiki sustave razvija se prema načelima paradigme otvorenog koda i dana je zajednici na besplatno korištenje.

Curriculum Vitae

Markus Schatten was born on September 27th in Vienna, Austria. Due to his parents business commitments he lived in Vienna, Budapest, Laxemburg, Zagreb and Povile. He finished the elementary school "Ivana Mažuranića" in Novi Vinodolski, an graduated from Prva riječka hrvatska gimnazija gymnasium highschool. Afterwards he graduated at the Faculty of Organization and Informatics in information systems with the BSc thesis "Klasifikacija dvodimenzionalnih slika ljudskih lica pomoću neuronskih mreža" (Classification of 2D Face Images using Neural Networks) and mentor doc. dr. sc. Miroslav Bača in 2005. In the same year the thesis was awarded with the deans degree. In 2008 he finished his MSc thesis "Zasnivanje otvorene ontologije odabranih segmenata biometrijske znanosti" (Developing an Open Ontology of Selected Segments of Biometrics) with mentor prof. dr. Miroslav Bača, and co-mentor Mirko Čubrilo. He authored and co-authored more than 20 professional and scientific papers on various conferences and journals. At the current time he is a research and teaching assistant at the chair for formal and applied foundations of information sciences at the Faculty of Organization and Informatics where he teaches 'knowledge management, programming, database theory and logic programming courses.

Home address: Zagrebačka 27
42000 Varaždin

Typesetting for this thesis was done with LaTeX 2_ε[1].

[1] LaTeX 2_ε which is an extension of LaTeX. LaTeX a collection of macros for TeX. TeX is a trademark of the American Mathematical Society. The macros used in formatting this thesis were written by Dinesh Das, Department of Computer Sciences, The University of Texas at Austin, and extended by Bert Kay, James A. Bednar, and Ayman El-Khashab. Additionally changed by Markus Schatten to fit the format of the Faculty of Organization and Informatics.