

Ontology Transformations Between Formalisms

Petr Aubrecht

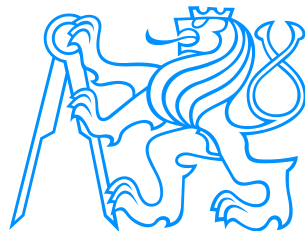
dissertation thesis

Study Programme: Electrical Engineering and Informatics

Branch of Study: Artificial Intelligence and Biocybernetics

2005

Supervisor: Prof. RNDr. Olga Štěpánková CSc.
Supervisor specialist: Doc. Ing. Zdeněk Kouba CSc.



The Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Cybernetics
Prague, Czech Republic

Acknowledgement

I would like to express my sincere gratitude and appreciation to my advisors Prof. RNDr. Olga Štěpánková CSc. and Doc. Ing. Zdeněk Kouba CSc. for providing me with the opportunity to work in the research in data mining and knowledge management, for their expert guidance and mentorship, and for their encouragement and support at all levels.

I would like to thank my colleagues, who helped me to start this thesis, for their advises and critiques. I would like to emphasise Kamil Matoušek and Monika Žáková.

Finally, I would like to thank my family for their love, support, and especially for my wife Věra's patience.

Brief Contents

Contents	vii
List of Figures	xi
List of Tables	xiii
1 Thesis Objectives	1
2 Introduction	3
3 Historical Background – From Chess Players to Ontologies	5
4 Ontologies	15
5 Ontology Transformations Between Formalisms	55
6 SumatraTT	99
7 Conclusion	125
Bibliography	127

BRIEF CONTENTS

Contents

Contents	vii
List of Figures	xi
List of Tables	xiii
1 Thesis Objectives	1
2 Introduction	3
2.1 Used Notation	4
3 Historical Background – From Chess Players to Ontologies	5
3.1 From Chess Players to Computing Machines	5
3.2 Arise of Artificial Intelligence	7
3.3 Expert Systems	9
3.4 The Cyc Project	10
3.5 Raise of Internet	11
3.5.1 Searching on the Internet	12
3.5.2 Google	13
3.6 Semantic Web	13
4 Ontologies	15
4.1 History of Ontologies	15
4.1.1 Semantic Web	18
4.2 Current Definitions of Ontology	20
4.3 Formal Definitions	21
4.3.1 Concepts and Relations	22
4.3.2 Ontology and Ontology Formalism	22
4.3.3 Simple Formalisms	25
4.3.4 Running Example	26
4.4 Overview of Formalisms	28
4.5 Frame-Based	28
4.5.1 Conceptual Graphs	29
4.5.2 KIF	29
4.5.3 Ontolingua	31

CONTENTS

4.5.4	OCML	32
4.5.5	CycL	33
4.5.6	UML, E-R Diagrams	34
4.6	Description Logics	35
4.7	Designed for Semantic Web	37
4.7.1	Topic Maps	37
4.7.2	SHOE	38
4.7.3	XOL	38
4.7.4	RDF	39
4.7.5	RDF Schema	40
4.7.6	DAML-ONT	42
4.7.7	OIL	42
4.7.8	DAML+OIL	42
4.7.9	OWL	43
4.8	Non-ontology Formalisms	45
4.9	Software Support	47
4.9.1	Editors	47
4.9.2	Engines	48
4.10	Upper Ontologies	49
4.10.1	Cyc	49
4.10.2	WordNet	50
4.10.3	Wikipedia	50
4.10.4	SUMO	51
4.10.5	CRM	52
4.11	Others	53
4.11.1	Dublin Core Metadata Initiative	53
5	Ontology Transformations Between Formalisms	55
5.1	Motivation	55
5.1.1	CIPHER Project	56
5.2	Migration within One Formalism	57
5.3	Known Approaches	59
5.3.1	Mapping Approach	59
5.3.2	Pivot Approach	59
5.3.3	Layered Approach	60
5.3.4	Family of Languages	60
5.3.5	Separated Worlds	60
5.4	Software Support	61
5.4.1	Ontolingua	61
5.4.2	Generic Frame Protocol	62
5.4.3	Open Knowledge Base Connectivity	62
5.4.4	Chimaera	63
5.4.5	Further Tools	63
5.5	Formal Definitions	64

5.6	Generalised Ontology Formalism	66
5.6.1	Evolution of Relations	67
5.6.2	Generalised Ontology Formalism Definition	68
5.6.3	Gates	71
5.6.4	Formalism-Specific Ontology	71
5.6.5	Mapping Engine	72
5.6.6	Models of Selected Formalisms	73
5.6.7	(Un)Informed Transformation	75
5.6.8	Solution for Untransformable Parts of Ontology	79
5.6.9	Operations	79
5.6.10	Common Problems Solved	80
5.6.11	Subproperty Problem	81
5.6.12	Instance of Instance	84
5.6.13	Restriction Handling	85
5.6.14	Visualisation	86
5.7	Implementation	90
5.8	Results	93
5.8.1	Testing Environment	93
5.8.2	Measurements	93
5.9	Conclusion	97
6	SumatraTT	99
6.1	History of SumatraTT	99
6.1.1	SumatraTT 1.0	99
6.1.2	SumatraTT 2.0	100
6.1.3	Graphical Interface for Data Preprocessing	102
6.2	Features	103
6.2.1	Basic Concepts	103
6.2.2	Advanced Features	104
6.2.3	Stream design	105
6.2.4	AutoDocumentation	107
6.2.5	Available Modules	109
6.2.6	Amount of Processed Data	112
6.3	SumatraTT and Data Transformation Tasks	113
6.3.1	Visualisation	115
6.3.2	Some additional modules	116
6.3.3	Practical Application in Data Preprocessing	116
6.4	SumatraTT in Knowledge Management	117
6.4.1	Testing Modules	117
6.5	Implementation of Generalised Ontology Formalism in SumatraTT	121
6.5.1	GOF Implementation	121
6.5.2	Module Conversion	121
6.5.3	Example	121
6.5.4	Background Knowledge of Transformation	121

CONTENTS

6.6 SumatraTT Summary	122
7 Conclusion	125
Bibliography	127

List of Figures

3.1	Analytical Engine	6
3.2	Colossus helped to break Nacist secret military codes	7
3.3	First American Computer	8
3.4	CERN, Tim Berners-Lee: figure from WWW proposal	11
4.1	Aristotle's ten basic categories (in leaves)	15
4.2	Conceptual graph from 13 th century by Ramon Lull	16
4.3	Linnaean taxonomy	17
4.4	Conceptual graph: Tom believes [Mary wants [to marry a sailor]]	18
4.5	Semantic web: superimposed semantic networks from multiple descriptions	19
4.6	Architecture of Semantic web from December 2000	20
4.7	Running example as a taxonomy	27
4.8	Running example in a semantic network	27
4.9	Running example in a conceptual graph	30
4.10	Running example in UML	34
4.11	Hierarchy of semantic web formalisms	37
4.12	Upper concepts in Cyc	49
4.13	Wikipedia article count from January 2001 to December 2004	50
4.14	Upper concepts in SUMO	51
4.15	Upper concepts in CRM	52
5.1	A schema of a part of CIPHER tools	58
5.2	Separated world of ontology formalisms	61
5.3	Ontolingua, translation from one formalism into multiple ones	62
5.4	The architecture of the Common Lisp implementation of OKBC.	63
5.5	Running example using <i>is-a</i> and <i>has-a</i> relations	68
5.6	Examples of GOF relations	69
5.7	Sample Ontology in GOF	70
5.8	Variants of arrays in properties	70
5.9	Partially defined properties in DLs	70
5.10	Running example using the generalised ontology formalism	71
5.11	GOF Gates	72
5.12	Piece of ontology to be mapped	72
5.13	FSO of frames	74
5.14	FSO of RDFS	75

LIST OF FIGURES

5.15 FSO of OWL	76
5.16 Ontology in GOF without (a) and with FSO (b)	77
5.17 Informed transformation	78
5.18 Uninformed transformation	79
5.19 Subproperty in GOF	81
5.20 Impossibility to draw a graph with subproperty in RDFS	81
5.21 Subproperty in the running example	82
5.22 First (bad) solution of the subproperty problem	82
5.23 Ontology with subproperties	84
5.24 Sample Ontology in GOF	84
5.25 Translated ontology	85
5.26 OWL restriction model in GOF	85
5.27 Tree (1.5D) visualisation of ontology	87
5.28 2D visualisation of ontology (TouchGraph)	88
5.29 3D visualisation of ontology (Wilmascope)	89
5.30 Prefuse output in detail	91
5.31 Prefuse output – overview	91
5.32 Hypergraph output in detail	92
6.1 SumatraTT 2.0 screenshot	101
6.2 SumatraTT 2.0 Structure	107
6.3 Support for documentation in transformation schema	108
6.4 Project documentation in HTML format	109
6.5 Example of scripting in SumatraTT	110
6.6 SumatraTT 2.0 XML processing	111
6.7 Scatter plot matrix for STULONG data – dependencies between pairs of selected attributes: weight, systolic pressure, diastolic pressure, choles- terol, nicotine level and time.	114
6.8 3D graph with column split wrt. classification	116
6.9 A tree of class inheritance generated by the Apollo2dot module	118
6.10 An entry page of the PATExporter output	119
6.11 HTML pages generated by the PATExporter module with an index of classes	119
6.12 An instance with an image attached with a list of properties	120
6.13 Ontology processing in SumatraTT	122
6.14 An ontology migration between the OWL and Prolog formalisms	122
6.15 An ontology migration with verification	123

List of Tables

4.1	Comparison of a formal language and ontology definitions	23
4.2	Ontology Formats	29
4.3	Set of constructors of Description Logics	35
5.1	Example of mapping rules	73
5.2	Mapping rules for frames	74
5.3	Mapping rules for OWL	77
5.4	Summary of differences between the knowledge models of RDF and Protégé-2000	80
5.5	Warm-up times for Apollo and OWL engines	94
5.6	Whole informed transformation times	95
5.7	Uninformed transformation times	95
5.8	Time of ontology loading without FSO ($\phi_e = \emptyset$)	96
5.9	Time of ontology loading with FSO ($\phi_e \neq \emptyset$)	96
5.10	Informed transformation times (without loading and saving)	96
5.11	Mapping times	97

Abstract

This thesis describes a platform for sharing ontologies in the world of various formalisms, editors, inference and query engines. For seamless cooperation of future systems it is necessary to bridge differences in design and purpose between various ontology sources.

Ontology formalisms tend to be rather complex and from certain point their semantics become unclear. Historically older formalisms incorporate general Lisp expressions as parts of ontologies, newer ones started from a simpler model, but the gradually increasing express power concurrently introduces requirement of computation into the ontology.

The ontology structure is clearly defined in simple formalisms like topic maps. Adding computations means decreasing this clarity and the ontology is locked up within the particular language.

This work defines a formal definition of ontologies in order to clearly separate different kinds of information. Further it introduces a framework for ontology sharing and transformation together with a solution how to handle the untransformable parts of ontologies. The framework is based on a Generalised Ontology Formalism (GOF), a graphical language consisting of a set of concepts and six relations. There are defined rules for transformation between GOF and generally used formalisms as well as all the accompanying definitions and theorems.

1 Thesis Objectives

As internet was growing, intelligent processing of the available information and searching relevant data within it became necessary. The need of computer programs with “common sense” also meets the problems of limited use of expert systems. Programs have to know the context of their work in order to provide meaningful results.

Ontologies are used for description of the world. An example of a long-term project developing ontologies is Cyc, aiming to describe all the human background knowledge and allow computers to think “with human-level breadth and depth of knowledge.”

Ontologies are created in various languages, depending on the requests of the domain and available tools suitable for the given domain. One group of such languages is based on Lisp, developed especially for artificial intelligence. Languages in this group usually use frames, introduced by M. Minsky in 1975. In the other group there are languages using description logics constructs and XML for storage.

As the term “language” is used in several meanings, “language” will be called the way of encoding for persistence purposes. Language at this level is an ordered set of symbols. A description at a higher level will be denoted as “formalism.” Several formalisms (e.g. RDF) allow encoding in more than one language (XML or N3).

Unfortunately, a conversion of ontologies between formalisms is a rather difficult problem. Formalisms are used being mutually incompatible in their constructs and often include procedural constructs to express features, dependencies, or restrictions in the knowledge base.

Goals

The main objective of this thesis is to provide a framework for transformation of ontologies between various formalisms. In order to achieve this major objective, the following goals had to be accomplished:

- Currently available definitions of *ontology* do not provide sufficient formal basis for making analyses, comparisons, and conversions of ontologies. Hence, a more formal definitions of ontology from syntactical point of view and definitions of further formal terms like *formalism* are necessary to be introduced.
- Formal definition of ontology transformation has to be defined making use of the developed formal definition of ontology.
- Demands on properties of such transformations need to be specified.

1 Thesis Objectives

- A generalised formalism, which will allow expressing meta-models of all common ontology formalisms, needs to be designed and the respective meta-models of individual ontology formalisms shall be expressed by means of this generalised one.
- Individual transformations making use of the generalised ontology formalism and respective ontology formalism meta-models shall be designed.
- As the existing ontology transformations have different expression power, it is not always possible to achieve conversion without losing some information. An analysis of ontology transformations needs to be done with respect to the natural requirement to lose as few information as possible.
- The whole designed framework needs to be implemented and verified on existing publicly available ontologies. Proper candidates are large upper ontologies SUMO and Cyc.
- Finally, the developed methodology needs to be evaluated.

2 Introduction

The main aim of this work in the beginning was making selected huge ontologies available in a different ontology formalism than they were originally created. A manual rewriting became too demanding and consequently a request for automatic translation appeared. Later in this work it will be shown, which approach has been selected, what problems were solved, and in which domains the solution can be applied.

The second main task done in this work is a formal framework for defining *ontology formalisms*. The term *ontology* is exactly defined as a means to allow comparing ontology formalisms and to identify transformable parts of ontologies.

In chapter 3, a short introduction to a history of artificial intelligence (AI) and mainly evolution of knowledge representation will be presented. At the end of this chapter it will be clear, that AI suffers from a limited scope of understanding of the context of their domain and there is a need for ontologies as a background information about the world out of the scope of the system.

The content of this thesis is divided to three parts.

The first part is formed by chapter 4 and presents ontologies as structures for expressing knowledge and assertions about the world. Section 4.3 introduces formal definitions, which will be used through this work. Terms *ontology* and *formalism* are defined and a new term *ontology grammar* is established.

A list of common formalisms is described in chapters 4.4 – 4.8 including few non-standard. A description for several important formalisms is provided in the previously introduced definitions. In order to simplify understanding of structures of the formalisms there is developed a running example. A simple ontology is encoded in most of the presented formalisms. A short description of upper ontologies is given in 4.10.

Chapter 5 represents the second part of this thesis and is concerned with ontology transformations between formalisms. After presenting the state-of-the-art in the domain, an original solution is presented together with samples of transformations between some formalisms introduced in the previous chapter. The solution is based on a formalism called *Generalised Ontology Formalism* (GOF) and consists of a set of concepts and six relations between them.

GOF is used in two roles – as a mediator in transformations of ontologies between formalisms and as a means to describe meta-models of considered formalisms. There are presented two methods of such transformation, their difficulties and solutions. Also a set of operations on ontologies in GOF is specified. At the end of the chapter 5 results of experiments are presented.

The third main part of the thesis is presented in chapter 6. It describes the software framework called SumatraTT, its purpose, internal structure, and role in the ontology processing. Section 6.4 is dedicated to experimental modules in knowledge management.

SumatraTT serves as a documentation platform for experiments and verifications of various ideas. The supported formalisms and operation are represented as modules in SumatraTT and a transformation can be composed from them.

Final chapter 7 summarises all the achievements of this thesis.

2.1 Used Notation

In the whole thesis the following notation is used:

Rel/n	relation with arity n ($Rel/n \subseteq \mathcal{C}^n$)
$rel(x_1, x_2, \dots, x_n)$	member of Rel/n : $\langle x_1, x_2, \dots, x_n \rangle \in Rel/n$
Ω	ontology, see definition 4.4 at page 24
\mathcal{F}	ontology formalism, see definition 4.5 at page 24
Ψ	formalism grammar, see definition 4.3 at page 23
\mathcal{C}	set of concepts
$\mathcal{C}_{\mathcal{F}}, \mathcal{R}_{\mathcal{F}}, \mathcal{S}_{\mathcal{F}}$	set of concepts, relations, struct. restrictions of formalism \mathcal{F}
$\mathbb{S}_{\mathcal{F}}, \mathbb{A}_{\mathcal{F}}$	languages used to specify additional restrictions and actions of formalism \mathcal{F}

The Generalised Ontology Formalism, introduced in chapter 5.6, uses six kinds of relations. Their graphical representation is listed in the following table:

Name	Graphics	Description
instanceOf	$\dashv\circ$	decreasing the abstractness of a concept
subclassOf	\blacktriangleright	specialisation relation between a more general and a more specific concepts
has-domain	\blacktriangleleft	domain of a property
has-range	$\dashv\sqcap$	range of a property
propertyOf	$\dashv\llcorner$	an assignment of a value to an instance of a property domain
has-value	\rightarrow	a particular value of a property

3 Historical Background – From Chess Players to Ontologies

The history of science unwinds in waves between enthusiasm and hope that it is possible to achieve superior goals and comebacks to rudiments to start over with more realistic aims. Alchemists during the era of Rudolph the II spent a lot of time on producing gold and constructing perpetuum mobile before research in chemistry and physics. During the time the goals lower to realistic level. At the moment capabilities of science and technology meet with demands and the results are presented, new inadequate requirements are imposed.

This cycling is true especially for the domain of Artificial Intelligence. In the beginning, it promised to solve any problem. In 1957 Herbert Simon stated, that machines already think, learn, and create. The year after he predicted that within 10 years a computer will become chess champion and prove a new important mathematical theorem.

In fact, at the beginning of the 21th century there are some partial successes, but it is more than clear, that there are even more problems than ever expected. Before the problem targeted in this thesis (a interoperability between knowledge representations) is described, allow me to make a short historical introduction.

In the historical overview a special attention will be payed to information representation. Some other topics, like evolutionary computing, are omitted.

3.1 From Chess Players to Computing Machines

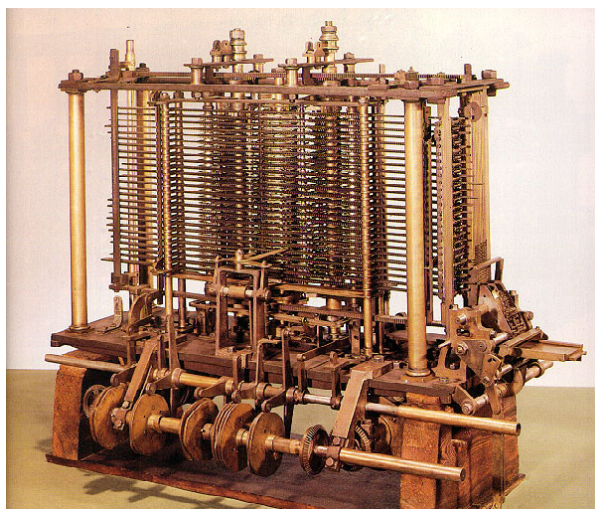
A specific area of great expectations is a construction of thinking machines outperforming human intelligence. The first presented “success” was a chess machine known as “The Turk,” invented in 1769 by Baron Wolfgang von Kempelen, a nobleman of Presburg, although in this case intelligence of the machine was in fact intelligence of the dwarf closed inside. It took more than two hundred years machines to defeat human chess players. In 1988, HITECH defeated Grandmaster Arnold Denker and on May 11 1997, DEEP BLUE defeated Garry Kasparov, reigning world champion, in a classical chess match. But in 19th century, the computers only started to learn to compute. . .

In 1837, Charles Babbage (1792–1871) designed an Analytical Engine (AE),¹ a mechanical digital computer. The idea of a mechanical computer had its predecessors – from abacus 7000 years ago to Blaire Pascal’s gear-based mechanical adding machine in 1645. The advantage of the AE was its general engine programmable by punched cards. Ada Byron, Countess of Lovelace, described AE in 1842 and wrote several programs

¹A simulator of the Analytical Engine can be now tried on site <http://www.fourmilab.ch/babbage/>.



(a) Charles Babbage



(b) Analytical Engine

Figure 3.1: Analytical Engine

for it, becoming the first computer programmer in the world.² In the description she expressed her scepticism about intelligence of machines: “The Analytical Engine has no pretensions to *originate* anything. It can do *whatever we know how to order it to perform.*” (her italics)

1940 An invention of digital electronic computers is dated to World War II simultaneously in three countries. Alan Turing’s team built in 1940 in the UK electromechanical Heath Robinson intended to decipher German messages. When Germans switched to more sophisticated code using their Enigma machine, the deciphering became too slow. Turing
1943 started development of Colossus using faster vacuum tubes. It was completed in 1943. Thanks to the machines developed at Bletchley Park and initial help of Polish cryptanalysts, Britain was able to read any German message during almost whole World War II. Their information was even much complete than German units itself. Whole work done in Bletchley Park was declassified in the mid-1970s, revealing this part of history of computers.

1941 Konrad Zuse invented the first programmable computer Z-3 in 1941 in Germany. His machine was using floating-point numbers and in 1945 he developed the first high-level programming language Plankalkul.

1942 In the United States John Atanasoff and Clifford Berry assembled the first electronic computer between 1940 and 1942 at Iowa State University, but the project was early abandoned. The most famous machine ENIAC, known as the first general-purpose, electronic, digital computer, was designed by Drs. Eckert and Mauchly. Construction

²The programming language ADA is named in Ada Byron’s honour.

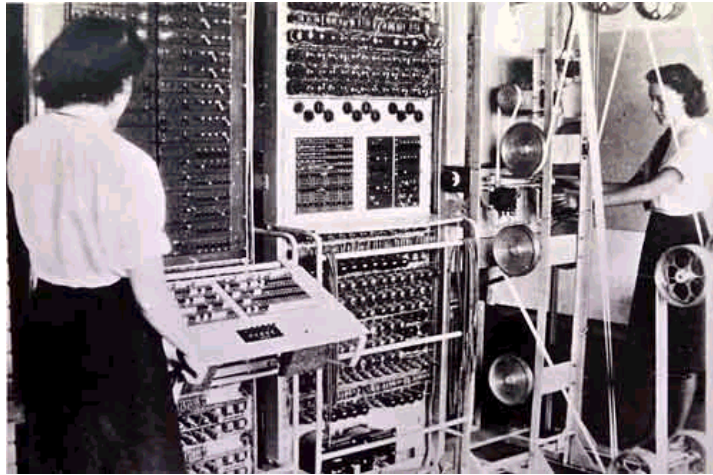


Figure 3.2: Colossus helped to break Nacist secret military codes

of the ENIAC was completed in the fall of 1945. Its very first application was to solve atomic energy problems for the Manhattan Project. 1945

3.2 Arise of Artificial Intelligence

The start of what we now recognise as artificial intelligence was in 1943 a work of Warren McCulloch and Walter Pitts, who proposed a model of artificial neurons. Not only they showed that any computable function (according to the Turing's theory of computation) can be computed by a network of connected neurons; they also suggested that suitably defined network could learn. Later, in 1949, Donald Hebb demonstrated a simple updating rule for modifying the connection strengths between neurons. 1943 1949

Early 1950s show a rapid research in AI. Alan Turing proposed his test providing a satisfactory operational definition of intelligence [51]. Claude Shannon (1950) and Alan Turing (1953) wrote chess programs for von Neumann-style conventional computers. Marvin Minsky and Dean Edmonds build the first neural network computer in 1951, SNARC, which simulated a network of 40 neurons. Their position was a rather hard as the research in artificial intelligence has been seen as frivolous and not serious enough. They had limited or no access to computers, Minsky's Ph. D. committee had an objections whether the area should be considered as mathematics, etc. 1950 1953 1951

The official birthplace of the AI field became Dartmouth College, where John McCarthy moved to. In the summer of 1956, he organised a two-month workshop to bring together U.S. researchers interested in automata theory, neural nets, and the study of intelligence. McCarthy convinced Minsky, Claude Shannon, Nathaniel Rochester, Trenchard from Princeton, Arthur Samuel from IBM, and Ray Solomonoff and Oliver Selfridge from MIT. Although all of them had ideas and in some cases particular applications such as checkers, the most interesting result was the first reasoning program. 1956

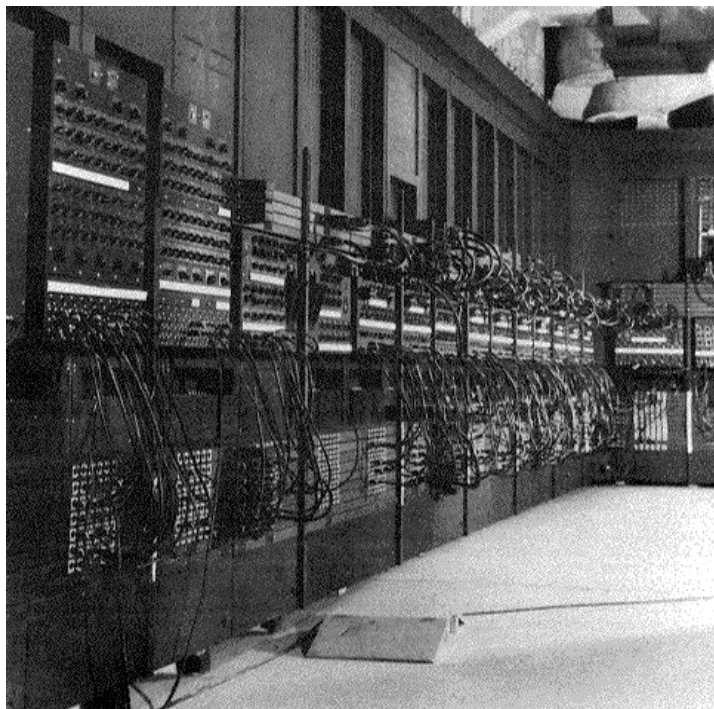


Figure 3.3: First American Computer

Allen Newell and Herbert Simon from Carnegie Tech³ implemented working reasoning program Logic Theorist which was designed to prove theorems from the famous logical work *Principia Mathematica* by Alfred North Whitehead and Bertrand Russell.

The Dartmouth workshop had two main results: it got altogether all the major figures and they agreed to adopt McCarthy's new name of the field: *Artificial Intelligence*.

1958 An important year is 1958 when McCarthy defined the high-level language Lisp (LIST Processing) based on λ calculus dating from 1936. This programming language is still in use in AI. McCarthy stressed representation and reasoning in formal logic. The article *Programs with Common Sense* was probably the first paper about logic as a method of representation of real-life knowledge and not just the subject matter of the program. He presented his hypothetical program Advice Taker, which embodied the central principles of knowledge representation and reasoning: formal, explicit representation of the world, reasoning, and ability to manipulate the representation with deductive processes.

Minsky, on the other hand was interested in really working programs and concentrated on limited domains (which became known as microworlds). James Slagle presented SAINT program for solving simpler closed-form integration problems.

1961 Newel and Simon went on after their Logic Theorist and in 1961 presented General Problem Solver (GPS), which was designed to embody the "human thinking" approach to solve problems. GPS generated and tested solutions under the guidance of heuristics supplied by the programmers. The criticism of GPS is that the program's "intelligence"

³Carnegie Tech is now Carnegie Mellon University.

is entirely coming from the programmer (mainly via the heuristics).

Frank Rosenblatt introduced perceptron in 1962 and proved his famous perceptron convergence theorem supporting his learning algorithm. Robustness and parallelism of a large number of elements representing an individual concept have been showed by Winograd and Cowan in 1963. 1962
1963

An important problem has been addressed in 1965 – a natural language communication. Two best-known systems are Eliza (human therapist) and Parry (human paranoiac). Both communicated in English, but they just syntactically manipulated sentences prepared in advanced.⁴ Of course, these early programs contained little or no knowledge of the subject of interview. Similar problems appeared in machine translations. 1965

3.3 Expert Systems

The greatest success of micro-world approach is a type of systems known as Expert Systems. In 1969, Bruce Buchanan, Ed Feigenbaum, and Joshua Lederberg were finding molecular structure from mass spectrometer measurements. Their program DENDRAL was the first system separated the inference engine from a knowledge base. The same concept was repeated in MYCIN with two differences: the knowledge was acquired by interviews with experts, and the calculus of uncertainty was involved. The MYCIN system outperformed junior doctors and was as good as some experts. 1969

A number of languages have been developed for knowledge representation. In Europe and Japan became popular Prolog (PROgramming in LOGic), using powerful theorem-proving technique known as *resolution* and which was first implemented in 1973. In declarative description of a problem is closer to the human’s way of thinking. Other recent work includes the development of languages for reasoning about time-dependent data such as “the invoice was paid yesterday”. These languages are based on tense logic, a type of logic that permits statements to be located in the flow of time. (Tense logic was invented in 1953 by the philosopher Arthur Prior at the University of Canterbury, New Zealand.) 1973

A different approach represented more structured frames of Marvin Minsky in 1975. A frame represented a particular object and collected facts about it. Frames were arranged into large taxonomic hierarchies. 1975

One of the most ambitious intelligent systems research was the Japanese project “Fifth Generation” in 1981. Prolog was selected as a base language. This project funded with big budget JPY50 billion ended in 1992 without any commercial success except some spin-offs (e.g. KLIC). 1981

Minsky and Papert’s *Perceptron* paper in 1969 caused 17 years of disinterest in neural networks. They returned to the scene in 1986. A back-propagation learning algorithm from 1969 was reinvented and successfully applied to many learning problems in computer science and psychology. 1986

⁴In fact Parry passed Turing’s test, because psychiatrists, who were asked, were often unable to say, whether they communicate with Parry or human paranoiac.

Approximately at the same time it showed that a successful expert system demands more than to buy a reasoning system and fill it with rules. The most serious drawback of expert systems is that they do not have “common sense”, they don’t know what they are for, nor limits of their applicability, nor how their recommendations fit into larger context. If MYCIN were told that a patient who has received a gunshot wound is bleeding to death, the program would attempt to diagnose a bacterial cause for the patient’s symptoms.

3.4 The Cyc Project

1984 The problem of expert systems with the “common sense” (CS) can be overcome by a huge knowledge base. With this idea in mind Douglas Lenat started Cyc (from “encyclopedia”), the largest project (experiment) in symbolic Artificial Intelligence, in 1984 in Microelectronics and Computer Technology Corporation in Texas. The initial budget was US\$ 50 million and the project continues up to these days.

The goal of Cyc is to build up the largest knowledge base containing a substantial portion of CS for future generation of expert systems. Lenat expects that Cyc needs about 100 million assertions to start to learn itself from written text.

The “common sense” means all the knowledge humans use in everyday life, including the simplest information which seems trivial to adult people. The CS include intuitions, expressions of ordinary language, foundational beliefs, and axioms. The complexity of the task expressed Terry Winograd, when he remarked “It has long been recognised that it is much easier to write a program to carry out abstruse formal operations than to capture the common sense of a dog.”

The Cyc knowledge base is divided into many (currently thousands of) “microtheories”, each of which is essentially a bundle of assertions that share a common set of assumptions; some microtheories are focused on a particular domain of knowledge, a particular level of detail, a particular interval in time, etc.

The information is so far entered by “cyclists” manually, by reading newspapers and magazines, encyclopaedias, advertisements etc. especially looking for background knowledge assumed to be known to readers. Lenat has predicted that in the early years of the new millennium, Cyc will become “a system with human-level breadth and depth of knowledge”.

At the present time, the Cyc knowledge base contains nearly two hundred thousand terms and several dozen hand-entered assertions about/involving each term. They are formulated in the language CycL, which is based on predicate calculus and has a syntax similar to the Lisp programming language.

A part of the Cyc project is publicly available under LGPL (open source) license as OpenCyc.

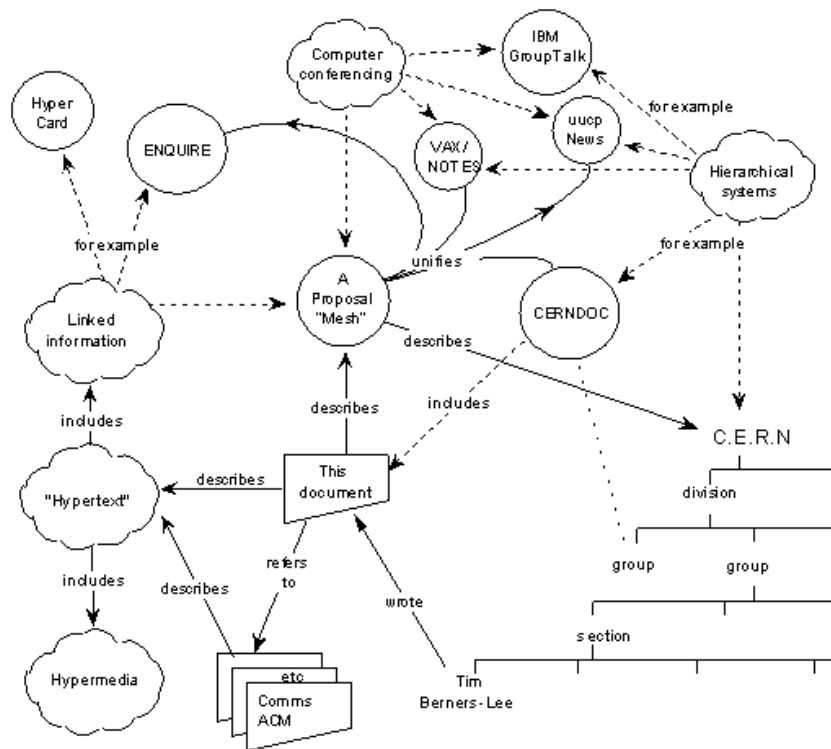


Figure 3.4: CERN, Tim Berners-Lee: figure from WWW proposal

3.5 Raise of Internet

A big challenge to AI is amount of information available on the Internet. For example text mining became an important part of AI research. Finding of particular piece of information in huge amount of text makes it possible to effectively use Internet.

The first packet network ARPANET was constructed in order to build a military research network that could survive a nuclear strike by *Defense Advanced Research Project Agency* (DARPA) in late 60s. It interconnected four nodes (universities) and nobody expected the boom with exponential growth to nowadays hundreds of millions hosts.

An important step in practical use of interconnected computers was introduction of WWW – World Wide Web by Tim Berners-Lee from CERN in 1989 (see figure 3.4).

1989

The enormous amount of information today publicly available on the Internet allowed the automatic crawling and gaining knowledge from the text. Unfortunately the automatic processing is dependent on natural text understanding, which is not yet at the level allowing its practical use.

A solution to the text understanding problem could be a semantic description of the content of the pages by authors, declaring meaning and context of whole page or pieces of the text. Unfortunately, currently the case with HTML shows, that in most cases authors are not enough educated and irresponsible and produce invalid code (not

following the standard), although they can at least use automatic validators. There must be some motivation, some advantage of generating a valid structure and an investment of an extra effort to markup text.

Success in web search engines can serve as a motivation to produce valid pages with markup, mainly in Google. Authors spend an important part of the design of the whole web site and page authoring by searching the right keywords to succeed in the Google search engine. The goal is to be highly ranked and the pages should appear at the first places in the search result, where are easily found by customers.

This kind of motivation could help automatic crawlers – if they process only valid pages and use meta-information, authors will be more precise and supply a correct semantic tags.

There is one more aspect – web services as a part of a business to business communication. This business area becomes more and more important and the whole industry, including web auctions, markets, marts etc. become quickly dependent on publicly available web services. Many providers will offer various services and an application development becomes a service integration – just combine available services and data transformation.

The whole evolution shows, that everything available through web should be accompanied with semantic information, what it is and how it can interact with other things.

This is the very basic idea of the *Semantic Web*.

3.5.1 Searching on the Internet

The rapid growth of Internet quickly caused a necessity of a universal search service. It was not possible to keep overview over all servers and their content.

A support for searching is included into the HTML standard, one of the cornerstones of the Internet, by its meta-information. The tag intended for searching is the **keywords** meta-tag: `<META name="keywords" content="list,of,keywords">`.

As most of the pages on the Internet have no keywords attached, searching by keywords is impossible, degrading the original idea.

1994 Processing of the content of web pages became the crucial task for search engines. In the middle 90s a group of web services appeared (e.g. Yahoo! since 1994, Altavista, Seznam). They looked for pages containing exactly the given words. For successful query it is necessary to enter the words punctually. They primarily process the text of the web pages as an unstructured information and rank only numbers of words found in the page. They do not try to understand the content in any way.

This approach is weak in several ways – first it is impractical for languages using many forms of one word to express e.g. tenses. This problem is addressed by national companies in corresponding countries. For example jyxo.cz is able to search all forms of the given words.

A way how to find more results, is a use of synonyma.

The biggest problem users currently face, it is the need to use the right keywords witch appear on the searched page. If the query contains general words, the amount of

results is enormous; if the query is too specific, there are no pages found. In both cases, the answers is unusable.

In this situation there can be ontologies employed, allowing a change of a generality of the given words. The pages will be searched in a more human way, by “meaning” of the query. Such access can be seen as an attempt to “understand” the text of the indexed pages.

3.5.2 Google

Google is probably the biggest phenomenon nowadays in web searching. Most of the English speaking Internet users search by the Google and it gives the most valuable results.

Google, a start-up dedicated to providing the best search experience on the web, was founded in September 1998 by Larry Page and Sergey Brin, building on three years of research as computer science Ph.D. candidates at Stanford University. Traffic of their experimental search engine has been growing at a rate of 50 % per month since Google’s inception. In June 7, 1999 Google announced it has received a \$25 million in equity funding. 1998

The Company’s name is derived from “Googol,” number 1 followed by 100 zeros, and reflects the immense amount of information available on the Internet. Google’s mission is to organise the world’s information, making it universally accessible and useful. Currently (end of 2004) Google indexes approximately 8 billion pages.

Unfortunately, as Google is a commercial company, their algorithms and use of ontologies is not clear (or at least not credible enough). More than on consistent usage of ontologies, Google rather relies on statistics and a set of heuristics called PageRank, a patent-pending, measure of the importance of web pages.

As an example of a search web already employing ontologies (although not so famous) can be mentioned www.clusty.com, offering limited search to specific areas, called clusters. Another attempt is www.hotbot.com.

3.6 Semantic Web

A similar idea to the Cyc one is the idea of a semantic web. It is more concentrated on the content of web pages than expert systems – it allows software agents to find the required information and perform simple actions like ordering some goods.

The basic idea of the semantic web has been expressed in article *The Semantic Web* by Tim Berners-Lee, James Hendler and Ora Lassila in 2001: “A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities.” [6] 2001

Machines cannot understand and interpret the meaning of the information in natural language – in which is most information on the web today. For the semantic web the information must be in a precise, machine-interpretable form.

3 Historical Background – From Chess Players to Ontologies

Internet in the future should be used by *Semantic Web agents* automatically processing shared data. They are going to lookup services available, help searching the requested information or even interoperate between home electronic appliances. In the business domain, the semantic software will be able to integrate services smoothly.

The semantic web is mentioned in more detail in section 4.1.1 for it is crucial for the ontology development in the latest years.

Techniques used for semantic web agent communication are studied by the discipline of AI known as multiagent systems (MAS). Agents in MAS can acquire their own private knowledge, which has to be later shared with other agents. To be able to communicate, agents share common dictionary/ontology, which provides a common language together with basic knowledge in the agent community.

4 Ontologies

Ontologies are used in computer science for establishing a common, precisely expressed knowledge, which provides a platform for communication among a group of either people or computers. The form of the representation had various forms during history and also today there is a wide range of different approaches to this problem.

4.1 History of Ontologies

The problem how to explicitly represent knowledge was first addressed by philosophers. The further text concentrates on the evolution of the structures designated for knowledge representation, especially hierarchies of concepts. Other issues like reasoning are considered to be out of the scope of this text.

In the 5th century B.C. Socrates discussed a proper definition of fundamental subjects like Truth, Beauty, Virtue, and Justice – as a result, Socrates himself professed his ignorance. His student Plato established the subject of epistemology, the branch of philosophy that deals with the nature, origin and scope of knowledge; he defined knowledge as justified true belief.

Plato’s student Aristotle (428 B.C.) concentrated on a more practical problem of knowledge representation. He started with constitution of terminology in many areas – logic, physics, biology, psychology, politics, ethics, economics, and others. These terms are used up to today, e.g. category, quality, quantity, metaphor, hypothesis, and many others. Figure 4.1 shows Aristotle’s ten basic categories (shown at the leaves).

The first graphical notation with representation of knowledge – today known as a semantic network – appeared in the thirteen century in a book *On Aristotle’s Categories* by the philosopher Ramon Lull. Figure 4.2 has been lent from [45].

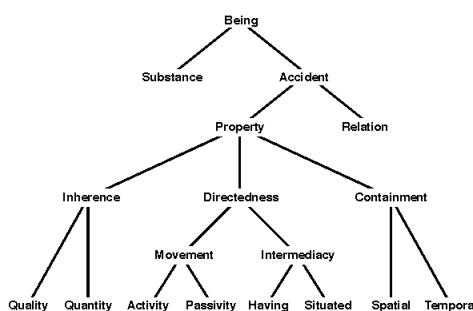


Figure 4.1: Aristotle’s ten basic categories (in leaves)

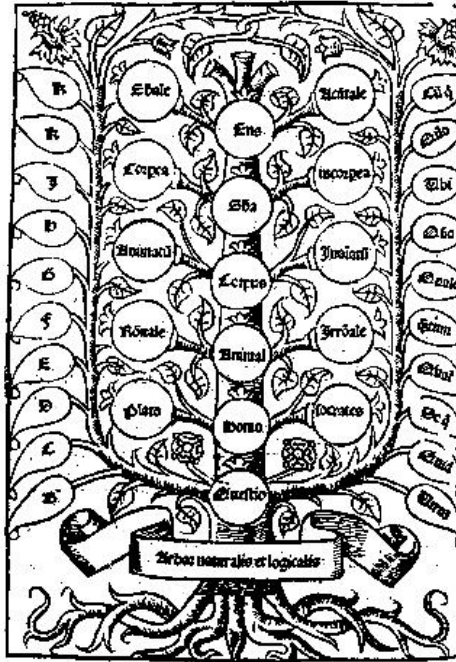


Figure 4.2: Conceptual graph from 13th century by Ramon Lull

Wilhelm Leibniz (1646–1716) wanted to establish mathematical foundation of mental processes. He assigned prime numbers to primitive concepts and multiplied them to derive composite concepts. This formalism is able to express only conjunction, Leibniz never found a way how to represent all the rules of inference and logical operators.

Ontologies, even in their simple form, were used long before artificial intelligence appeared. A simple form of ontology – taxonomy – has been used by Swedish scientist Carl Linnaeus (1707–1778, later Carl von Linné) for classification of plants (see figure 4.3). The Linnaean taxonomy is a base for taxonomies used by biologists today. It classifies living things within a hierarchy.

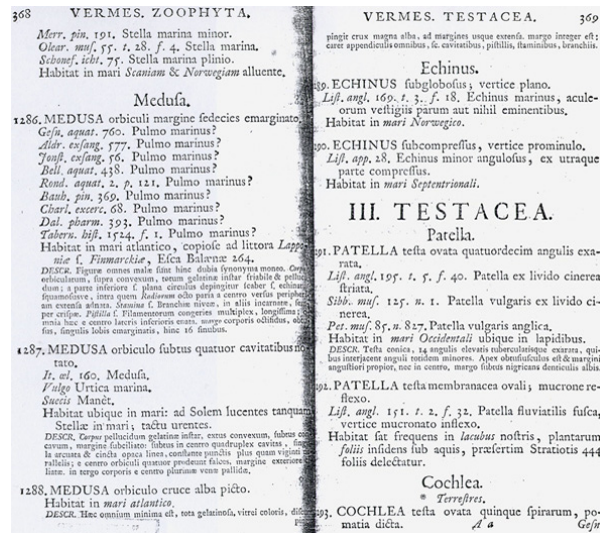
In 1956 Richard H. Richens of the Cambridge Language Research Unit defined “semantic nets” for machine translation of natural languages. Ross Quillian’s introduced in his Ph.D. thesis (1968) a term “semantic network” as a way of talking about the organisation of human semantic memory, or memory for word concepts. The network has a form of a map with points or nodes representing individual concepts and labelled links. The links express semantic types *is-a*, *instance-of*, *part-of* or general (user) semantics. Examples of semantic networks can be found in section 4.1.1, e.g. figure 4.5. Semantic networks are still used for example in Wordnet (<http://www.cogsci.princeton.edu/~wn/>) for representing synonyms, antonyms, hyponyms, and meronyms. Also wikipedia.org, which is often referenced in this work, is based on semantic networks.

Marvin Minsky proposed *frames* to represent facts and structure of some object as a record in [33] in 1975¹. The usage of frames were demonstrated on spatial imagery

¹There are three references to this article – in years 1974, 1975, and 1981. In fact, the article appeared



(a) Carl von Linné



(b) Systema naturae

Figure 4.3: Linnaean taxonomy

and linguistic understanding. Frames are defined as data structures for representing a stereotyped situation. Frames are organised into network of nodes and relations, where “top levels” are fixed, represent things that are always true about the supposed situation. The lower levels have many terminals – *slots* that must be filled by specific instances or data. Together with the slots there can be defined conditions an assignment must meet; they can require a terminal assignment to be of a particular type or an object of sufficient value; more complex conditions can specify relations among the things assigned to several terminals. Facets were later added to express more detailed description of a slot (default values, restrictions, etc.).

Description logics (DL, previously called terminological logics) started with a motivation of providing a formal foundation for semantic networks. The first DL implementation KL-ONE grew out of Ron Brachman’s thesis in 1977. In 1983, Ron Brachman introduced T-box for defining terms and A-box for making assertions. Using prime integers for T-boxes and Leibniz’s products for inheritance (a subclass of two classes is represented by a product of the two corresponding numbers) introduces lattice. Lattice is still used in many knowledge-management applications (e.g. FCA).

Around 1990 there were several implementation based on description logics – LOOM (KL-ONE style system, MacGregor, 1987), Classic (Bogida, 1989), Kris (Baader, 1991), etc. On the basis of description logics were built languages targeted to the semantic

three times: first, as an MIT-AI Laboratory Memo 306 in June, 1974, later in *The Psychology of Computer Vision*, P. Winston (Ed.), McGraw-Hill in 1975, shorter version in J. Haugeland, Ed., *Mind Design*, MIT Press, 1981, and for the last time in *Cognitive Science*, Collins, Allan and Edward E. Smith (eds.) Morgan-Kaufmann, 1992.

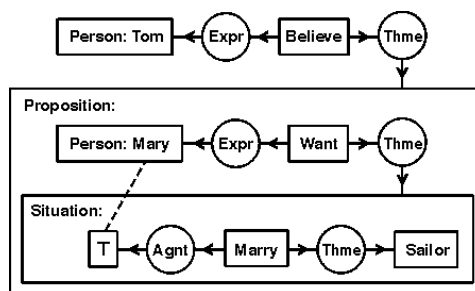


Figure 4.4: Conceptual graph: Tom believes [Mary wants [to marry a sailor]]

web; they are described in the next section (RDF in 1997 and others).

John F. Sowa's conceptual graphs are examples of a visual notation for predicate calculus [46]. Conceptual graph is an extension to C. S. Pierce's² existential graph with features adopted from AI and linguistics. An example of a conceptual graph is in figure 4.4. The figure represents a sentence Tom believes [Mary wants [to marry a sailor]].

In the last decade, the emergence of large amount computers joined in one huge network allowed efficient cooperation of software agents searching a particular piece of information or a service. Such agents have to apply their own model of world describing terms used by sources accessed. This plan leads to the semantic web, presented in the next section.

4.1.1 Semantic Web

The first ideas, how interconnected computers should effectively manage information, provided Tim Berners-Lee in the original proposal of the WWW in 1989.³ He discussed problems of loss of information about complex systems in CERN and suggested a solution based on a distributed hypertext system. The solution to this problem was HyperText Markup Language (HTML).

The HTML language became accepted as a language for World Wide Web. As the number of documents is extremely large and continues to grow rapidly, there is a necessity of a precise search with relevant answers. It requires a classification of the content comprehensible to machines. HTML did not provide a way how to semantically markup the content. A promising way how to impose an understanding of the content of the web by computers is the idea of a semantic web. The content of the web pages will be described in terms of standardised ontologies and thus any search engine will be able to find a relevant information.

Early attempts to use metadata to add and extract meaning to the web page appeared

²Because of the nature of the writings of Pierce (1839–1914) most of his work was never published during his lifetime, although much of it is available in collected volumes – e.g. 8 volumes “The collected papers of Charles Sanders Pierce,” Harvard University Press, 1931–1958.

³source: <http://www.w3.org/History/1989/proposal.html>

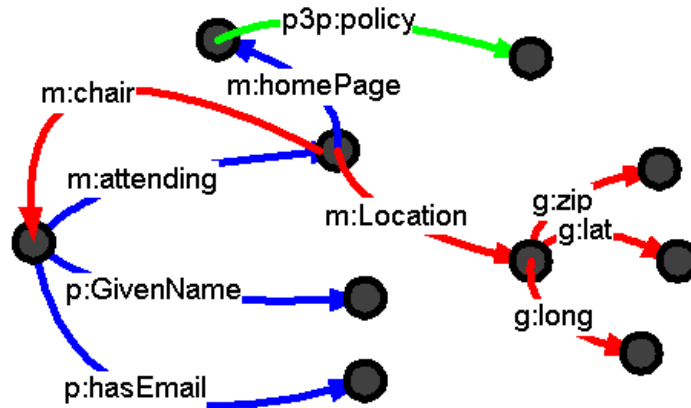


Figure 4.5: Semantic web: superimposed semantic networks from multiple descriptions

in 1993 in the HTML draft 1.2 in element `link` (pointer from this document to another one) with attributes `rel` (describing a type of the link) and `rev` (reverse – how the target document relates to the current one). Because there were no advantages of using this element, no pages were marked by it.

Another attempt was Simple HTML Ontology Extensions (SHOE) in 1995, trying to add ontology into web page, which was never widely used.

As a formalism for adding semantics to the page was later chosen the Resource Description Format (RDF) with its extension RDF Schema (RDFS). It allowed to relate any Uniform Resource Identifier (URI) with any other URI or literal. In the December 2000, Tim Berners-Lee presented the idea of the semantic web⁴ – the early layered schema is in figure 4.6. The main role had to play the layer RDF+RDFS using RDFS formalism with its “very wide interoperability”. For all the layers were presented existing technologies able to serve the tasks of the layer.

Using RDFS as a formalism for *Ontology vocabulary* led to problems, because the RDFS standard is too general (weakly defined) to be reasonably processed. Instead of using the RDFS formalism as the only language, it was quickly replaced by its successors.

The substitution of the RDFS layer in the schema should be understood as a challenge to define a method, how to migrate ontologies written in different formalisms. Single layers then could be developed independently and composed according to needs of an application.

The first formalism widely used for developing ontologies for the semantic web was DARPA Agent Markup Language (DAML). The DAML Program formally began with a kickoff meeting in August 2000 in Boston. In December the same year it joined with European project OIL, which added some procedural features (restriction on properties, set operations). The Defense Advanced Research Projects Agency in the February 2004 announced that the World Wide Web Consortium approved DAML as an international standard.

⁴source: <http://www.w3.org/2000/Talks/1206-xml2k-tbl/slide1-0.html>

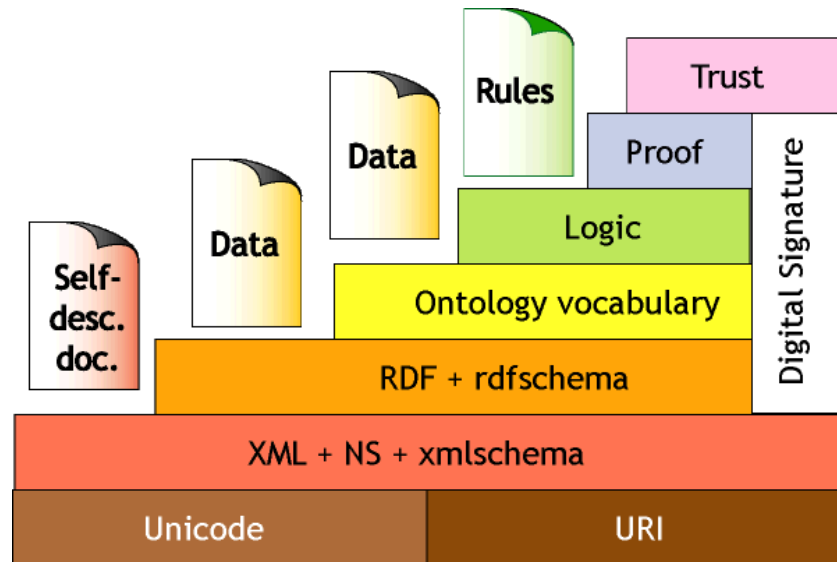


Figure 4.6: Architecture of Semantic web from December 2000

Web-Ontology (WebOnt) Working Group proposed a new language Web Ontology Language (OWL) based on DAML+OIL. The working group consists of 51 commercial and public sector organisations, research projects/labs, and invited experts. The goal of the languages were shared ontologies, ontology evolution, ontology interoperability, inconsistency detection, balance of expressivity and scalability, ease of use, XML syntax, and internationalisation.

All the formalisms based on RDF are used up to today. The RDFS is a base for the RSS channels, providing simple information about new article on information webs. DAML serves as a formalism for the biggest repository of ready-to-use ontologies at daml.org. OWL is the newest one and it is chosen for new projects, it seems that in the future ontologies will converge to OWL.

The standards connected with the semantic web idea are currently covered by the W3C World Wide Web Consortium with their motto: *Leading the Web to Its Full Potential...*

4.2 Current Definitions of Ontology

The term *ontology* itself has several definitions, which are mutually incompatible. Originally the term came from philosophy. In the computer science, especially in knowledge management the term *ontology* was used for a structure representing knowledge of a system about the world around.

Wikipedia (<http://en.wikipedia.org/wiki/Ontology>) In philosophy, ontology, is the most fundamental branch of metaphysics. It is the study of being or existence as well as the basic categories thereof. It has strong implications for the conceptions of reality.

Webster's Revised Unabridged Dictionary (1913) That department of the science of metaphysics which investigates and explains the nature and essential properties and relations of all beings, as such, or the principles and causes of being.

WordNet (r) 2.0 *n* : the metaphysical study of the nature of being and existence

In computer science, this word got a different sense. Its most important role is to provide a common set of concepts for information interchange:

Gruber, 1996, [16] An ontology is a explicit specification of a conceptualisation.

Borst, 1997, [7] An ontology is a formal specification of a shared conceptualisation.

Sowa, 2000, [45] Ontology defines the kinds of things that exist in the application domain.

Wikipedia, 2004

http://en.wikipedia.org/wiki/Ontology_%28computer_science%29

In computer science, an ontology is the attempt to formulate an exhaustive and rigorous conceptual schema within a given domain, a typically hierarchical data structure containing all the relevant entities and their relationships and rules (theorems, regulations) within that domain. The computer science usage of the term ontology is derived from the much older usage of the term in philosophy, where it means the study of being or existence as well as the basic categories thereof. See ontology (philosophy).

The difference between the Gruber's and later Borst's definitions is in the requirement of using a formal language for expression of the specification. It can be shown on the case of the CRM ontology – it is written as a text without any formal definition and thus conforms to the Gruber's definition, but not the Borst's.

The definitions provided restrict the domain of ontology research in knowledge representation in contrast to the philosophical “study of being”. They are general enough to allow wide range of variants, requiring only presence of *concepts*. They do not anticipate any particular features the form should have.

There exist multiple formalisms following those definitions, but different in their capabilities. In order to find a common platform allowing mutual translation between ontologies and their formalisms, there is necessary to introduce a more formal definition. Such definition is provided in the next chapter.

4.3 Formal Definitions

For the investigation of transformations there is a need of an exact mathematical definition of the term *ontology*. The definition decreases the level of abstraction of the term and allows us to draw conclusions about ontology transformations. Together with the definition of ontology there is presented a definition of an ontology formalism.

The terms used for definitions will be demonstrated in section 4.3.3 from the simplest formalisms to more complex. Descriptions of formalisms in section 4.4 contain definitions of real-life formalisms.

4.3.1 Concepts and Relations

Concept is every object in the domain of discourse, either abstract or generic idea generalised from particular instances. Concepts in the further text will be used as a basis for developing other definitions. It will be represented by a unique literal.

A set of concepts will be denoted \mathcal{C} . In the ontology in figure 4.1 all names of classes are concepts.

To connect concepts into structures, n -ary relations will be used. A relation in this case is denoted by an n -ary predicate (a truth-valued function of n variables) $\text{rel}(x_1, x_2, \dots, x_n)$, where $x_1, x_2, \dots, x_n \in \mathcal{C}$. In the following text we use the abbreviation *Relation/n* to express a set of relations with n arguments (for example *SubclassOf/2*):

Definition 4.1 (Relation with arity n).

$$\begin{aligned} \text{Rel}/n &\subseteq \mathcal{C}^n \\ \text{rel}(x_1, x_2, \dots, x_n) &\Leftrightarrow \langle x_1, x_2, \dots, x_n \rangle \in \text{Rel}/n. \end{aligned}$$

A set of all relations will be denoted \mathcal{R} . For example, the labels in figure 4.5 belong to the \mathcal{R} set.

An example of usage of the shortcut is a definition of a *subclassOf* relation:

$$\text{SubclassOf}/2 = \{\text{subclassOf}(x_1, x_2) \mid x_1, x_2 \in \text{Concepts}\}$$

All ontology formalisms, except the trivial ones, define restrictions. To define restriction, there is needed definition of a transitive closure of a binary relation:

Definition 4.2 (Transitive closure R^* of a binary relation R). *The transitive closure R^* of a relation R is defined by*

$$\begin{aligned} xRy &\Rightarrow xR^*y \\ xRy \wedge yR^*z &\Rightarrow xR^*z \end{aligned}$$

I.e. elements are related by R^ if they are related by R directly or through some sequence of intermediate related elements.*

4.3.2 Ontology and Ontology Formalism

Basically, ontology content can be divided into two basic parts: structural and procedural. The structural part is addressed far more often, for example by the Gruber's definition (explicit specification of conceptualisation, [17], chapter 4.2). We further divide the structural part into a set of concepts and a set of relations between the concepts. The procedural part consists of restrictions and actions.

Formal language definition	Ontology definition
symbol	concept
alphabet (finite set of symbols)	set of used concepts, \mathcal{C}
string (finite sequence of symbols)	ontology, Ω
language (set of strings)	ontology formalism \mathcal{F}
grammar (V, T, P, S)	ontology grammar Ψ
language generated by grammar L(G)	formalism generated by grammar $\mathcal{F}(\Psi)$

Table 4.1: Comparison of a formal language and ontology definitions

The definition of ontology in this work was inspired by a common definition of a formal language. Definition terms are compared in the table 4.1. Newly introduced terms are emphasised.

Every formalism works with a limited set of terms, which it uses. For example, the most common term is a “class.” Others terms involve property or slot, value, string etc. A set of all these terms for a particular formalism will be denoted $\mathcal{C}_{\mathcal{F}}$.

Very similarly, every formalism defines a set of relations (like is-subclass-of); it will be denoted $\mathcal{R}_{\mathcal{F}}$.

Ontology formalisms are often designed in a context of some language. It can be either a common programming language (often Lisp) or a new special language (OIL). This language is then used to restrict relations between concepts or to express actions to be performed in some situation. Languages will be further denoted by a bold mathematical font: $\mathbb{S}_{\mathcal{S}}$ and $\mathbb{A}_{\mathcal{S}}$.

With these typographic conventions the ontology grammar can be defined:

Definition 4.3 (Ontology grammar). *Ontology grammar is a 5-tuple:*

$$\Psi = (\mathcal{C}_{\mathcal{F}}, \mathcal{R}_{\mathcal{F}}, \mathcal{S}_{\mathcal{F}}, \mathbb{S}_{\mathcal{F}}, \mathbb{A}_{\mathcal{F}})$$

where

- $\mathcal{C}_{\mathcal{F}}$ is a set of formalism concepts,
- $\mathcal{R}_{\mathcal{F}}$ is a set of formalism relations,
- $\mathcal{S}_{\mathcal{F}}$ is a set of structural restrictions on relations between ontology concepts,
- $\mathbb{S}_{\mathcal{F}}$ is a language to specify additional restrictions applicable in the formalism, and
- $\mathbb{A}_{\mathcal{F}}$ is a language to specify actions allowed in the formalism.

The definition of the ontology grammar clearly separates the structural part – $\mathcal{C}_{\mathcal{F}}$, $\mathcal{R}_{\mathcal{F}}$, and $\mathcal{S}_{\mathcal{F}}$ from the procedural part – $\mathbb{S}_{\mathcal{F}}$ and $\mathbb{A}_{\mathcal{S}}$.

The structural part will be the subject to transformation between ontologies in the next chapter.

4 Ontologies

The procedural part consists of two languages, which in many cases have power of the Turing Machine. It is well-known fact, that it is not possible to find a conversion between two general languages ([27], chapter 7.3).

Simple examples of grammars will be given in section 4.3.3. More complex examples will be given in sections describing formalisms OCML (section 4.5.4) and formalisms based on description logics (section 4.6).

Now, ontology and formalism can be defined:

Definition 4.4 (Ontology Ω).

$$\Omega = (\mathcal{C}, \mathcal{R}, \phi_{\mathcal{C}}, \phi_{\mathcal{R}}, \mathcal{S}, \mathcal{A})$$

where

- \mathcal{C} is a set of concepts (concept is represented by a literal),
- \mathcal{R} is a set of relations between concepts,
- $\phi_{\mathcal{C}}$ is a function $\phi_{\mathcal{C}} : \mathcal{C} \rightarrow \mathcal{C}_{\mathcal{F}}$,
- $\phi_{\mathcal{R}}$ is a function $\phi_{\mathcal{R}} : \mathcal{R} \rightarrow \mathcal{R}_{\mathcal{F}}$,
- \mathcal{S} is a set of restrictions, and
- \mathcal{A} is set of actions.

Definition 4.5 (Formalism). A formalism is a set of ontologies with common sets of formalism concepts $\mathcal{C}_{\mathcal{F}}$ and relations $\mathcal{R}_{\mathcal{F}}$.

Although there can exist a formalism not covered by any ontology grammar, we will mention only formalisms generated by grammars. Grammars can generate any formalism using concepts. The part of knowledge not expressible using concepts is known as *tacit* there exist no means how to express it.

These definitions will be applied to the formalisms analysed for the transformation purpose.

A relation \prec required for a lattice of formalism definitions will be defined in the following text. The \prec relation plays a similar role for formalism grammars as \subset for sets.

Definition 4.6 (Formalism grammar \prec relation). Two formalism grammars $\Psi_1 = (\mathcal{C}_{\mathcal{F}1}, \mathcal{R}_{\mathcal{F}1}, \mathcal{S}_{\mathcal{F}1}, \mathcal{S}_{\mathcal{F}1}, \mathcal{A}_{\mathcal{F}1})$ and $\Psi_2 = (\mathcal{C}_{\mathcal{F}2}, \mathcal{R}_{\mathcal{F}2}, \mathcal{S}_{\mathcal{F}2}, \mathcal{S}_{\mathcal{F}2}, \mathcal{A}_{\mathcal{F}2})$ are in \prec relation, if one can be fully described using constructs of the other, i.e. there exist injective functions between the pieces of the grammars:

$$\begin{aligned} \Psi_1 \prec \Psi_2 \quad \text{if} \quad & \exists \phi_c : \mathcal{C}_{\mathcal{F}1} \rightarrow \mathcal{C}_{\mathcal{F}2} \wedge \\ & \exists \phi_r : \mathcal{R}_{\mathcal{F}1} \rightarrow \mathcal{R}_{\mathcal{F}2} \wedge \\ & \exists \phi_s : \mathcal{S}_{\mathcal{F}1} \rightarrow \mathcal{S}_{\mathcal{F}2} \wedge \\ & \exists \phi_{\mathcal{S}} : \mathcal{S}_{\mathcal{F}1} \rightarrow \mathcal{S}_{\mathcal{F}2} \wedge \\ & \exists \phi_{\mathcal{A}} : \mathcal{A}_{\mathcal{F}1} \rightarrow \mathcal{A}_{\mathcal{F}2} \end{aligned}$$

Definition 4.7 (General and specific grammar). For two formalism grammars

$\Psi_1 = (\mathcal{C}_{\mathcal{F}_1}, \mathcal{R}_{\mathcal{F}_1}, \mathcal{S}_{\mathcal{F}_1}, \mathcal{S}_{\mathcal{F}_1}, \mathbb{A}_{\mathcal{F}_1})$ and $\Psi_2 = (\mathcal{C}_{\mathcal{F}_2}, \mathcal{R}_{\mathcal{F}_2}, \mathcal{S}_{\mathcal{F}_2}, \mathcal{S}_{\mathcal{F}_2}, \mathbb{A}_{\mathcal{F}_2})$:

– the general formalism grammar is

$\Psi_{1 \cup 2} = (\mathcal{C}_{\mathcal{F}_1} \cup \mathcal{C}_{\mathcal{F}_2}, \mathcal{R}_{\mathcal{F}_1} \cup \mathcal{R}_{\mathcal{F}_2}, \mathcal{S}_{\mathcal{F}_1} \cap \mathcal{S}_{\mathcal{F}_2}, \mathcal{S}_{\mathcal{F}_1} \cap \mathcal{S}_{\mathcal{F}_2}, \mathbb{A}_{\mathcal{F}_1} \cup \mathbb{A}_{\mathcal{F}_2})$ and

– the specific formalism grammar is

$\Psi_{1 \cap 2} = (\mathcal{C}_{\mathcal{F}_1} \cap \mathcal{C}_{\mathcal{F}_2}, \mathcal{R}_{\mathcal{F}_1} \cap \mathcal{R}_{\mathcal{F}_2}, \mathcal{S}_{\mathcal{F}_1} \cup \mathcal{S}_{\mathcal{F}_2}, \mathcal{S}_{\mathcal{F}_1} \cup \mathcal{S}_{\mathcal{F}_2}, \mathbb{A}_{\mathcal{F}_1} \cap \mathbb{A}_{\mathcal{F}_2})$

Then we can say $\Psi_{1 \cap 2} \prec \Psi_1 \prec \Psi_{1 \cup 2}$.

Theorem 4.1 (Formalism grammar lattice). A set of formalism grammars and \prec relation over grammars form a lattice:

The least upper bound (\top) is $\Psi_0 = (\{\}, \{\}, \{\}, \{\}, \{\})$.

The greatest lower bound (\perp) is Ψ_∞ – grammar generating all possible ontologies.

Theorem 4.2 (Formalism \prec relation). Two formalisms are in \prec relation if their grammars are in the \prec relation:

$$\mathcal{F}_1 \prec \mathcal{F}_2 \quad \text{iff} \quad \Psi_1 \prec \Psi_2$$

Theorem 4.3 (Formalism lattice). A set of formalisms and \prec relation over formalisms form a lattice:

The least upper bound (\top) is $\mathcal{F}(\Psi_0)$.

The greatest lower bound (\perp) is $\mathcal{F}(\Psi_\infty)$.

4.3.3 Simple Formalisms

Let us start with some definitions of simple formalisms – empty ontology and a set of concepts (minimal ontology). Both of these formalisms are subsets of all the further formalisms:

$$\Psi_0 = (\{\}, \{\}, \{\}, \{\}, \{\}) \quad (4.1)$$

$$\Psi_{set} = (\{item\}, \{\}, \{\}, \{\}, \{\}) \quad (4.2)$$

$$\Psi_{taxonomy} = (\{concept\}, \{Is-a/2\}, \{concept \text{ is-a } concept\}, \{\neg(x \text{ is-a}^* x)\}, \{\}) \quad (4.3)$$

Taxonomy is the second simplest case of conceptualisation. Its corresponding oriented graph is a forrest. For this purpose, a necessity of the root (\top) is omitted to allow multiple top-level concepts: The rule $\neg(x \text{ subclassOf}^* x)$ means, that there are no cycles in the corresponding oriented graphs (where concepts are vertices and relations are edges).

Very similar to taxonomy is a lattice. In addition to the taxonomy, lattice has top (\top) and bottom (\perp) symbols:

$$\Psi_{lattice} = (\{concept, \top, \perp\}, \{Is-a/2\}, \{concept \text{ is-a } concept, concept \text{ is-a } \top, \perp \text{ is-a } concept\}, \{\neg(x \text{ is-a}^* x)\}, \{\}) \quad (4.4)$$

In the taxonomy (and lattice) formalism, the $\mathcal{S}_{\mathcal{F}}$ set was used for the first time. It defines, how relations and classes can be used: $\{concept\ is-a\ concept, concept\ is-a\ \top, \perp\ is-a\ concept\}$ means, that the *is-a* relation can have *concept* as both left and right argument, but \top only as its left and \perp as its left argument.

An advantage of the lattice formalism is its simplicity. Lattice can be automatically generated, e.g. by the Formal Concept Analysis (FCA), [41].

The most simple formalisms can be easily compared:

$$\Psi_0 \prec \Psi_{set} \prec \Psi_{taxonomy} \prec \Psi_{lattice}. \quad (4.5)$$

As an example of a complex language, a part of OCML definition is provided here. The full definition will be specified in section 4.5.4 dedicated to the OCML formalism:

$$\begin{aligned} \Psi_{OCML} = & (\{class, instance, slot, literal, assignment, \dots\} \\ & \{subclassOf/2, hasSlot/2, assignment/3, \dots\}, \\ & \{Slot-type-validations, \dots\}, \\ & User-Defined-Restrictions, Actions) \end{aligned} \quad (4.6)$$

Similar definitions can be provided also for description logic based formalisms, topic maps, ER diagrams, and even for non-ontology formalisms like Java classes hierarchy, bookmarks, or filesystem. Selected formalisms will be examined in section 4.4 including formal specifications.

4.3.4 Running Example

In the further text there will be reviewed several formalisms available. Some of them will be investigated further and for them there will be given examples.

In order to emphasise differences between the ontologies, there will be given an example. Here, the example domain will be defined in English first:

There exist two kinds of humans: a man and a women. Each human has parents – mother and father.

There are two particular people – Adam and Eve, and they have two sons: Abel and Cain.

A part of the formalisms presented in this chapter are based on logic, so the following piece of code represents the same knowledge in Prolog:

```
human(X) :- man(X).
human(X) :- woman(X).

man(Adam). man(Abel). man(Cain).
woman(Eve).

hasfather(Abel, Adam). hasfather(Cain, Adam).
hasmother(Abel, Eve). hasmother(Cain, Eve).
```

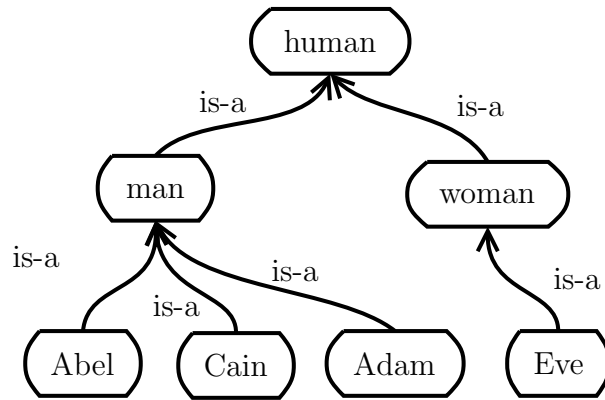


Figure 4.7: Running example as a taxonomy

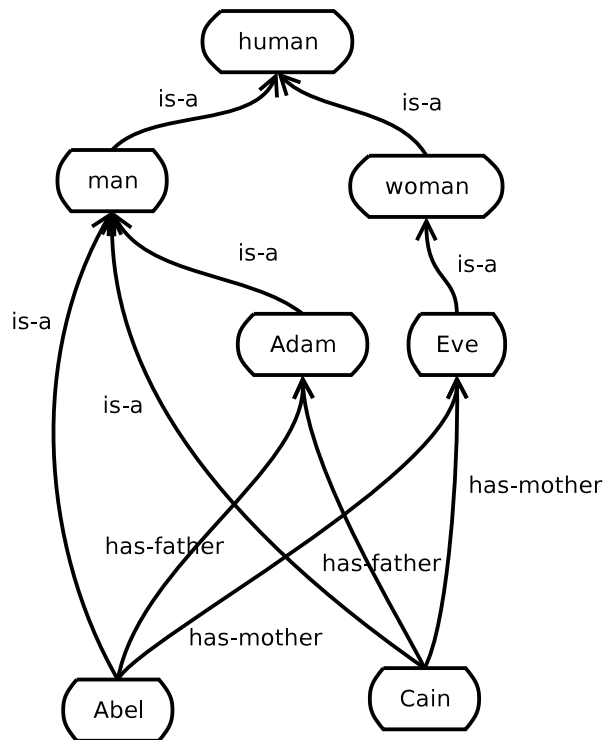


Figure 4.8: Running example in a semantic network

4.4 Overview of Formalisms

Before any transformation can be designed, the existing formalisms must be analysed. In the following sections set of formalisms will be shown and commented. Only formalisms usable for expressing ontology and thus interesting for ontology formalism transformation are chosen. Another overview with usage of formalisms is in [40] and [50].

Nowadays the usage of formalisms is split to two areas – artificial intelligence or knowledge management (especially with reasoners) and web description. The former area tends to use formalisms based on Lisp, while the other later uses formalisms rather based on RDF.

An example of Lisp-based ontology formalism is the OCML. It is used as a primary output of the Apollo editor, see sections 4.5.4 and 4.9.1.

On internet, the most frequently practically used formalism is RDF, e.g. RDF channels (used as a short description of recently published articles on some websites). These descriptions can be easily downloaded from multiple websites and offered to the user as an overview of currently available articles. However the RDF channels currently provide no information about classification of the content of the article. For that kind of description it is necessary to use a more “ontological” language. Such languages are RDFS and its successor OWL. These languages are able to describe the content in categorised terms.

Although RDFS and OWL are preferred by web-centric communities, these formalisms have their drawbacks. Their design does not follow conventions established in knowledge management community (e. g. they define no meta level – metaclasses for definition of classes; OWL DL tries to solve some of these issues).

There exist other formalisms with their own advantages and special features. These formalisms are targeted to specific domains like large taxonomies categorising Web sites, product catalogues, large information systems, domain knowledge in fields such as medicine or multiagent systems. An example of a special feature are microtheories introduced in Cyc. Microtheories attempt to follow the real world, where there exist multiple points of view and not just one general theory covering the whole human knowledge.

There are two basic foundations of ontology formalisms – logics and frames. Both are closely related, but in the frame-based systems a structure of terms is emphasised, while logics-based systems stress use of inference. Therefore the formalisms description is divided into two separated sections.

The table 4.2 shows a list of formalisms considered to be a subject of transformation presented in this work.

4.5 Frame-Based

Minsky’s so called “frames paper” [33] presented frames as a knowledge representation schema. It extended object-oriented programming technique, introduced by O. J. Dahn and K. Nygaard in their programming language SIMULA 67 in 1967. In this language were for the first time used both objects and classes, subclasses (usually referred to as inheritance).

Formalism	Year	Reference
Conceptual Graphs	1983	John F. Sowa, http://www.jfsowa.com/cg
KIF	1992	Genesereth and Fikes, [29]
Ontolingua	1992	T. R. Gruber, [16]
OCML	1995	E. Motta, [36]
Topic Maps	1991	http://topicmaps.it.bond.edu.au/
OKBC	1998	Chaudhri et. al, http://www.ai.sri.com/~okbc/
XOL	1999	http://www.ai.sri.com/pkarp/xol/
RDF	1997	www.w3.org/RDF/
RDF-S	2000	www.w3.org/TR/rdf-schema/
DAML	10/2000	www.daml.org
OIL		http://www.ontoknowledge.org/oil/
DAML+OIL	11/2001	http://www.ontoknowledge.org/oil/
OWL	07/2002	http://www.w3.org/2001/sw/WebOnt/

Table 4.2: Ontology Formats

In frame-based systems information is stored in frames, which consist of slots, describing the object. Particular values are assigned to the slots.

A slots consists of a set of facets. A value of the slot is one of the facets, other facets can contain default values or so called daemons. These daemons represent procedural information in frames. They can serve as value-checkers or general procedures. The daemons are used in a predefined situations like when is the slot value filled, changed or cleared. In this approach the range of slot (type) is checked by a corresponding daemon.

Frames become very popular thanks to their similarity to the object programming, which dominates computer design in last decade and thanks to easy mapping to the first-order logics.

There exist numerous ontology formalisms based on frames.

4.5.1 Conceptual Graphs

Ontology formalisms are not used only for building hierarchies of terms, but can keep information about particular objects and their interrelations. For example, conceptual graphs (CGs) are a system of logic based on the existential graphs of Charles Sanders Pierce and the semantic networks of artificial intelligence. They express meaning in a form that is logically precise, humanly readable, and computationally tractable. The CGs concentrate on the natural language processing and understandability to humans. An example of CG was shown in the introduction in figure 4.4.

4.5.2 KIF

Knowledge Interchange Format (KIF, [29]) defines itself as a communication format for interchange of knowledge among disparate computer systems written in different

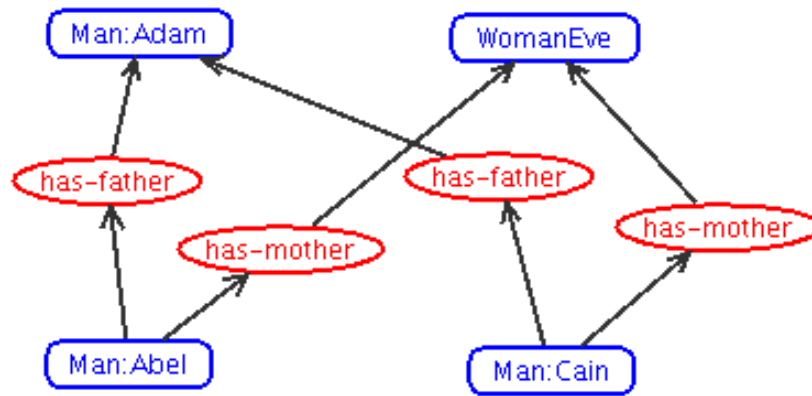


Figure 4.9: Running example in a conceptual graph

programming languages. The stress on a programmer-readable language facilitates the development of independent knowledge-manipulation programs. KIF is an extended version to the first-order predicate calculus and its syntax is based on LISP.

SUO-KIF

For the SUMO upper ontology (see section 4.10.4) a SUO-KIF language has been developed as a KIF extension. Here it will server as an example of usage of KIF. Here is a piece of the SUMO knowledge base:

```

(instance economyType BinaryPredicate)
(domain economyType 1 Agent) ;GeopoliticalArea or GovernmentOrganization
(domain economyType 2 EconomicAttribute)
(subrelation economyType attribute)

(documentation economyType "(&%economyType ?POLITY ?TYPE) means that the
&%GeopoliticalArea ?POLITY has an economic system of &%TYPE.")

(=>
  (economyType ?AGENT ?ATTRIBUTE)
  (or
    (instance ?AGENT GeopoliticalArea)
    (instance ?AGENT Organization)))

(=>
  (attribute ?AREA FormerSovietOrEasternEuropeanCountry)
  (economyType ?AREA CountryInTransition))

(economyType CzechRepublic CountryInTransition)

```

4.5.3 Ontolingua

Ontolingua [16] has been developed in 1993 as a system for describing ontologies in a form compatible with multiple representation languages. It provides forms for defining classes, relations, functions, objects, and theories. Ontolingua syntax and semantics are based on KIF.

A part of Ontolingua, so called *Frame Ontology*, defines a set of idioms that the system can recognise. The Frame Ontology contains second-order relations – a complete axiomatisation of classes and instances, slots and slot constraints, class and relation specialisation, relation inverses, relation composition, and class partitions.

The following example shows the running example in Ontolingua. There is also shown Ontolingua's ability to separate ontologies to parts using `in-theory`.

```
(in-theory 'FIRST-HUMANS)
(define-class HUMAN (?body) )
(define-class MAN (?body)
  :def (human ?body))
(define-class WOMAN (?body)
  :def (human ?body))
(define-relation FATHER (?parent ?child)
  :def (and (man ?parent)
            (human ?child)))
(define-relation MOTHER (?parent ?child)
  :def (and (woman ?parent)
            (human ?child)))
(define-instance ADAM (man)
  "The first people created by God"
  :slots
    ((father ABEL)
     (father CAIN)))
(define-instance EVE (woman)
  "First woman"
  :slots ((mother ABEL)
          (mother CAIN)))
(define-instance CAIN (man))
(define-instance ABEL (man))
```

An exact definition of Ontolingua (and formalisms based on LISP) is complicated by its parts written in LISP without any formalisation. The following example is from sample ontology:⁵

⁵source: <http://www-ksl.stanford.edu/knowledge-sharing/ontologies/html/mace-domain/mace-domain.lisp.html>

```
(define-instance GIMBAL-2-BASE (mechanical-component)
  :axiom-def (and (subcomponent-of GIMBAL-2-BASE GIMBAL-2)
    (= (orientation (reference-frame GIMBAL-2-BASE)
      (reference-frame PLANAR-MACE))
      (simple-rotation 3 180))
    (= (position (reference-point GIMBAL-2-BASE)
      (reference-point PLANAR-MACE))
      (* (- (buslength BUS-ASSY))
        (basis.vec (reference-frame PLANAR-MACE) 1))))))
)
```

The `:axiom-def` part of the definition uses LISP syntax and only patterns can be searched in order to acquire a structure of classes/instances. This feature prohibits using LISP-based formalisms in modern application, often using Java or C++ and thus unable to evaluate LISP expressions.

4.5.4 OCML

Operational Conceptual Modelling Language (OCML) was inspired by Ontolingua in its constructs. It adds mainly operational capabilities to ontologies, defines restrictions and action, which can be executed. An example of use of this feature is the time ontology in PhD thesis of Kamil Matoušek [28].

The OCML language was developed at the Knowledge Media Institute, The Open University, UK by E. Motta.

The OCML language was the main platform used in the CIPHER project (see section 5.1.1), which formulated aim of this thesis. Its usage allowed to develop an application *Story Fountain* presenting stories with ability to explore relations between its items. In fact, the Story Fountain is one of the first applications of Semantic Web.

Mechanisms provided by OCML allow expressing classes, instances, relations, functions, and rules (with both backward and forward chaining). OCML also implements general `tell&ask` interface as a mechanism for assertion of facts and examination of the content of a knowledge base.

For efficient reasoning, there are some extra logical mechanisms, such as procedural attachments. Procedures are pieces of LISP code. A definition of a class or a relation, it can consist of several parts. The parts `:iff-def`, `:prove-by` or `constraint` are evaluated during inference in order to cut the searched space. Moreover, there can be defined a `lispfun` part, which access internal OCML structures.

Practical results of this thesis use the OCML formalism as a primary output. For loading of OCML ontology, all the procedural part is not taken into account.

In OCML and similarly in all frame-based formalisms, an assignment (a specific slot is set to a particular value) is a relation with arity 3, as it connects the definition of the slot, the instance and the target value. Both the *Slot-type-validations* and *Actions* are Lisp functions connected to an event (assignment of a slot value and adding a new fact).

$$\Psi_{OCML} = (\{class, instance, slot, literal, assignment, relation\}, \quad (4.7)$$

$$\{SubclassOf/2, HasSlot/2, Assignment/3, TypeOf/2, SlotType/2\},$$

$$\{class subclassOf/2 class, class hasSlot/2 slot,$$

$$assignment/3(instance, slot, instance),$$

$$assignment/3(instance, slot, literal),$$

$$instance typeOf/2 class, slot slotType/2 class\},$$

$$\{Slot-type-validations, \dots\}, Actions)$$

```
(def-class HUMAN ()
  ((FATHER-OF :max-cardinality 1
              :type MAN)
   (MOTHER-OF :max-cardinality 1
              :type WOMAN)))

(def-class MAN (HUMAN) () )
(def-class WOMAN (HUMAN) () )

(def-instance ADAM man
  ((FATHER-OF (ABEL CAIN))))
(def-instance EVE woman
  ((FATHER-OF (ABEL CAIN))))
(def-instance CAIN man)
(def-instance ABEL man)
```

Practically, instead of parsing Lisp syntax of OCML, the software implementation in this thesis used an XML format of editor designed for OCML authoring – ApolloCH. During its development, there has been added functionality not available in OCML. An example of such feature is possibility to assing a class into a slot, not only instance or literal.

In the ApolloCH editor, the procedural part of ontology is not present.

A definition of ApolloCH formalism can be done by a simple extension of the OCML definition:

$$\Psi_{ApolloCH} = (\mathcal{C}_{OCML}, \mathcal{R}_{OCML}, \mathcal{S}_{OCML} \cup \{assignment/3(instance, slot, class)\}, \{\}, \{\}) \quad (4.8)$$

4.5.5 CycL

The Cyc project was already mentioned in the historical overview in chapter 3.4. Here will be described the language used for the project – CycL.

CycL is a frame language developed for the Cyc project. An important feature of the CycL formalism is division of a knowledge base to pieces called *microtheories*. Every

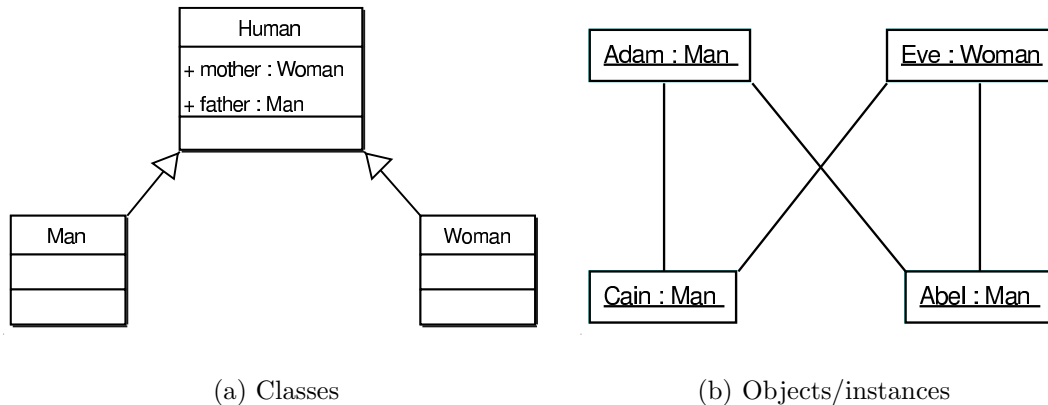


Figure 4.10: Running example in UML

assertion falls within at least one microtheory. The microtheories are used for better scalable knowledge base building, better inferencing and to cope with global inconsistency in knowledge base. At the huge scale the inconsistency is inevitable. Each microtheory is locally consistent. They are also good for handling divergence (different points of view, scientific theories, changes over time): “Tables are solid” – at a granularity usually considered by humans, tables are solid.

Microtheories can be dependant due to the `#$genlMt` inheritance predicate. The expression `(#$genlMt MT-1 MT-2)` means that every assertion which is true in MT-2 is also true in MT-1.

The running example expressed in CycL can be written in microtheory `#$AdamEveMt`:

```
(#$genls #Man #Human) \;
($genls #Woman #Human) \;
($isa #Adam #Man) \;
($isa #Eve #Woman) \;
($relationAllExists #biologicalMother #Human #Man)
($relationAllExists #biologicalMother #Human #Woman)
($biologicalMother #Abel #Eve)
```

4.5.6 UML, E-R Diagrams

In the computer design Unified Modelling Language (UML) and Entity-Relation (E-R) diagrams established as the two most commonly used standards.

E-R diagrams are used for design of structure of a relational database and define tables, attributes and relations between them.

The UML (<http://www.uml.org/>) allows application specification. Currently it is the world-leading technology with support of many software companies and many tools both commercial and free. It consists of several kinds of diagrams modelling different aspects of the application structure and behaviour.

For this text the structural (class and deployment) diagrams are important. The model in UML define not only simple parent-child relation, but also inheritance, aggregation, or composition (only the structure diagrams are important in this context).

UML has defined mapping to the most common languages, mainly Java and C++. An attempt of a special conversion of OWL ontology into Java source code can be found in [24].

4.6 Description Logics

Description logics play an important role in forming languages for the semantic web. A very short introduction to DL helps to understand details about issues of the particular languages.

Description Logics are a family of knowledge representation formalisms [20]. It represents information about classes and individuals and their description. It consists of *Concepts* – unary predicates denoting entities or classes, *Roles* – binary predicates denoting properties or relations, *Constructors* for concept expressions and *Individuals* denoting instances of classes.

The logics use various constructors to build complex classes from simpler ones. A list of the available constructors is in table 4.3. The choice of the set of constructors determine the expressivity and performance i.e. decidability of reasoning.

Construct	Syntax	Language
concept	A	FL^-
role name	R	
conjunction	$C \cap D$	
value restriction	$\forall R.C$	
existential quantification	$\exists R$	
top	\top	AL^*
bottom	\perp	
negation (C)	$\neg C$	
disjunction	$C \cup D$	
existential restriction	$\exists R.C$	
number restrictions	$(\leq nR)$	
collection of individuals	$\{a_1, \dots, a_n\}$	
role hierarchy	$R \subseteq S$	
inverse role	R^-	I
qualified number restriction	$(\leq nR)$	Q

Table 4.3: Set of constructors of Description Logics

A key feature of Description Logics is that they are logics, i.e. formal languages with well defined semantics.

The purpose of the semantics is to explicate the relationship between the language syntax and the intended model of the domain. A model consists of a *domain* ($\Delta^{\mathcal{J}}$) and an *interpretation function* ($\cdot^{\mathcal{J}}$). The domain is a set of objects and the interpretation function is a mapping from individual, class and property names to elements of the domain, subsets of the domain and binary relation on the domain. For example, for the class Human, $\text{Human}^{\mathcal{J}} \subseteq \Delta^{\mathcal{J}}$, for an individual Adam, $\text{Adam}^{\mathcal{J}} \in \Delta^{\mathcal{J}}$, and for the property fatherOf, $\text{fatherOf}^{\mathcal{J}} \in \Delta^{\mathcal{J}} \times \Delta^{\mathcal{J}}$.

Definition 4.8 (Description logic semantics). *Let $CON = C_1, C_2, \dots$ be a countable set of atomic concepts, $ROL = R_1, R_2, \dots$ be a countable set of atomic roles and $IND = a_1, a_2, \dots$ be a countable set of individuals. For CON, ROL, IND pairwise disjoint, $\mathcal{S} = \langle CON, ROL, IND \rangle$ is a signature. Once a signature \mathcal{S} is fixed, an interpretation \mathcal{J} for \mathcal{S} is a tuple $\mathcal{J} = \langle \Delta^{\mathcal{J}}, \cdot^{\mathcal{J}} \rangle$, where*

- $\Delta^{\mathcal{J}}$ is a non empty set,
- $\cdot^{\mathcal{J}}$ is a function assigning an element $a_i^{\mathcal{J}} \in \Delta^{\mathcal{J}}$ to each constant a_i ; a subset $C_i^{\mathcal{J}} \subseteq \Delta^{\mathcal{J}}$ to each atomic concept C_i ; and a relation $R_i^{\mathcal{J}} \subseteq \Delta^{\mathcal{J}} \times \Delta^{\mathcal{J}}$ to each atomic role R_i .

Any (basic) Description Logic is a subset of function-free first-order logic using at most three variable names in a formula. The representation is on the predicate level – no variables are used in the expressions. The mapping from FOL can be done effectively using lambda expressions ($\text{HUMAN} \equiv \lambda X. \text{HUMAN}(X)$).

The advantage of Description Logics over FOL is in its more efficient (decidable) inference procedures. The finer structure of knowledge can guide inference. For the simplest DL language, FL^- , which uses only conjunction, value restriction and existential quantification, it can be shown that reasoning is decidable in polynomial time.

A Description Logic theory is divided in two parts: the conceptual or terminological knowledge i.e. the definition of predicates (TBox) and the instance assertional knowledge i.e. assertion over constants (ABox). The reasoning is then done using a “knowledge base”.

Definition 4.9 (Knowledge base). *Fix a description language \mathcal{L} , a knowledge base Σ in \mathcal{L} is a pair $\Sigma = \langle T, A \rangle$ such that*

- T is the TBox, a finite, possibly empty, set of expressions of the form $C_1 \subseteq C_2$ where C_1, C_2 are in $CON(\mathcal{L})$. Formulas in T are called *terminological axioms*.
- A is the ABox, a finite, possibly empty, set of expressions of the form $a:C$ or $(a,b):R$, where C is in $CON(\mathcal{L})$, R is in $ROL(\mathcal{L})$ and a, b are individuals. Formulas in A are called *assertions*.

An example of a T-Box can be $\text{HAS-FATHER} = \exists \text{ParentOf.Man}$ and A-Box can be demonstrated on $\text{Man}(\text{Adam})$, $\text{Woman}(\text{Eve})$, or $\text{FatherOf}(\text{Adam}, \text{Abel})$.

Some description logics, such as ALC(D) [5] and SHOQ(D) [21] (DL into which OIL can be mapped completely) allow attributes to have values from so-called “concrete

domains,” which can contain entirely new kinds of values. These concrete domains are required to have their own, independent reasoners, which are then coupled with the DL reasoner.

The Description Logics were chosen as a base for ontology formalisms designed for the Semantic Web. The following section presents a list of formalisms practically used.

4.7 Designed for Semantic Web

A need of annotation of information available on web was described earlier (in sections 3.6 and 4.1.1). For this purposes have been developed numerous formalisms. A hierarchy of the most important ones is in figure 4.11.

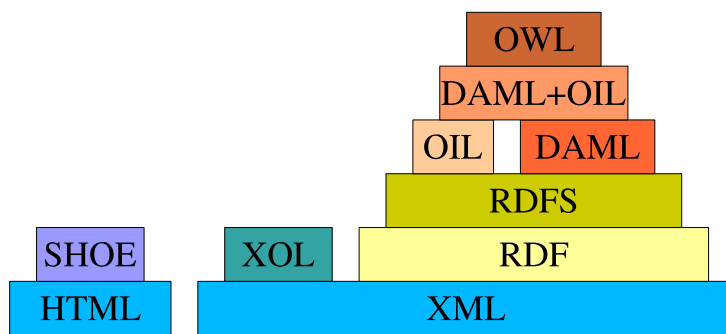


Figure 4.11: Hierarchy of semantic web formalisms

4.7.1 Topic Maps

Topic maps, recent ISO standard (ISO/IEC 13250), was developed by the HyTime (Hypermedia/Time-based Structuring Language) community. They annotate documents with conceptual information.

The terms which topic maps use, are *topic*, *association*, and *occurrences*. *Topic* represents the same kind of information as *concept* in this thesis. *Topics* are instances of *Topic types*, which are another topics, i.e. topic maps have no specialised primitive to distinguish classes and instances. *Association* expresses relation between *topics*. *Occurrences* store information, where (e.g. in an annotated text, on web) appears given topic.

Topic maps are very similar to semantic networks. They add the topic/occurrence axis to the knowledge base. In the context of this thesis, occurrences are not important and topic maps can be treated in the same way as semantic networks.

4.7.2 SHOE

The SHOE (Simple HTML Ontology Extensions) language defines primitives for ontology definition on web pages. Web documents written in HTML are enriched by annotations describing parts of the document.

The description is divided into two parts – *ontology*, kept in a separate HTML file and *annotation*, written directly inside the related web page.

SHOE operates with classes (called *categories*), which are organised in a simple *is-a* hierarchy. Instances of classes are always bound with a page or its part. Between instances of a given class are defined relations.

Thanks to simple design, the definition of the SHOE formalism consists of several rules:

$$\Psi_{SHOE} = (\{category, instance, relation\}, \{DefRelation/2, Related/2, InstanceOf/2\}, \{related/2(relation, instance, instance), instance instanceOf/2 category, category defRelation/2 category\}, \{\}, \{\}) \quad (4.9)$$

A part of the running example can be implemented as (the rest is obvious)

```
<DEF-CATEGORY NAME="Human" ISA="base.SHOEntity">
<DEF-CATEGORY NAME="Man" ISA="Human">
<DEF-CATEGORY NAME="Woman" ISA="Human">
<DEF-RELATION NAME="father-of">
  <DEF-ARG POS="1" TYPE="Human">
  <DEF-ARG POS="2" TYPE="Man">
</DEF-RELATION>

<INSTANCE KEY="http://www.heaven.org/Adam">
  <CATEGORY NAME="Man">
</INSTANCE>
<INSTANCE KEY="http://www.heaven.org/Eve">
  <CATEGORY NAME="Woman">
</INSTANCE>
<INSTANCE KEY="http://www.heaven.org/Abel">
  <RELATION NAME="father">
    <ARG POS=TO VALUE="Adam">
  </RELATION>
</INSTANCE>
```

4.7.3 XOL

The XOL formalism (XOL: An XML-Based Ontology Exchange Language) was developed in 1999 for a bioinformatics community. Crucial requirement was an XML format for object-oriented knowledge representation system.

The semantics of XOL are based on OKBC-Lite, which is a simplified form of the knowledge model for the OKBC (Open Knowledge Base Connectivity, see section 5.4.3). The design of the language is inspired by Ontolingua – the advantage is the XML storage instead of LISP.

Similarly to SHOE, XOL defines classes, slots and individuals (same as category, relation, instance). The explanation can be replaced by the running example:

```
<class>
  <name>Human</name>
</class>
<class>
  <name>Man</name>
  <subclass-of>Human</subclass-of>
</class>
<slot>
  <name>fatherOf</name>
  <domain>Man</domain>
  <slot-value-type>Human</slot-value-type>
</slot>

<instance>
  <name>Adam</name>
  <type>Man</type>
  <slot-values>
    <name>fatherOf</name>
    <value>Adam</value>
  </slot-values>
</instance>
<instance>
  <name>Abel</name>
  <type>Man</type>
  <slot-values/>
</instance>
```

The inspiration by OKBC brought into XOL several basic types (integers, real numbers, string, and booleans), collections (set, bag, list) and a set of built-in classes (THING, CLASS, etc.) and a slot DOCUMENTATION. It has also limited restrictions on slot values beside obvious slot type – there are called facets. There can be bounded cardinality, defined type of collection or bounded numeric value. An interesting facet is INVERSE allowing bidirectional relation between slots.

4.7.4 RDF

The first draft of RDF was released in October 1997. It was developed by the W3C for a creation of metadata describing web resources – RDF stands for *Resource Description*

Framework.

One of the goals of RDF is to be complementary to XML – to make it possible to specify semantics for data based on XML in a standardised, interoperable manner. The goal is to define a mechanism for resource description that makes no assumption about a particular application domain nor the structure of a document.

The RDF data model is based on semantic networks. The graph consists of tripples (statements) subject, predicate, and object; the predicate (edge) denotes a relation between the subject and object (vertices). Predicates and objects are entities that can be referred by URI (or URL in the WWW), literal, or blank node. Properties define specific aspects of a the entities.

RDF supports reification. Statements are resources and thus can be described in RDF. It allows to express properties of such statement like validity, reliability, e.g. formulas in higher order logics. The other side of this advantage is its undesirable consequences for inference. For this reason the latest standards (OWL Lite and OWL DL) do not allow reification.

There are two basic types of RDF storage languages – XML and N3 tripples (subject, predicate and statement). The common way is to share data in XML, the tripples are used in inference engines.

There are numerous tools and further standards. Within scope of this thesis RDF is interesting only as a base for further formalisms.

4.7.5 RDF Schema

RDF Schema (also RDFS) is an extension of the RDF and introduces basic constructs for definition of ontology: classes, properties, literals, resources, and corresponding relations. It is a simple language able to define hierarchies of classes of resources, instances using `type` attribute, and properties with a range and a domain (and subproperties). There are missing for example restrictions.

RDFS suffers from several problems. First, RDFS was originally defined as a set of syntactic constructs and logical foundation and semantics were added later. Second, some approaches are not standard. For example there is no boundary between classes and instances (there can be a class instance of another class, `rdf:type` property used in a chain); `XMLLiteral` is defined as instance of `Datatype` but a subclass of `Literal`, see w3c website or figure 5.14.

The running example helps to compare RDFS formalism and its successors, DAML-ONT, DAML+OIL, and OWL.

```
<rdfs:Class rdf:ID = 'Person' />
<rdfs:Class rdf:ID = 'Man' >
    <rdfs:subClassOf rdf:resource = '#Person' />
</rdfs:Class>
<rdfs:Class rdf:ID = 'Woman' >
    <rdfs:subClassOf rdf:resource = '#Person' />
</rdfs:Class>
```



```

<rdfs:ObjectProperty rdf:ID = 'father'>
    <rdfs:domain rdf:resource='#Person' />
    <rdfs:range rdf:resource='#Man' />
</rdfs:ObjectProperty>
<rdfs:ObjectProperty rdf:ID = 'mother'>
    <rdfs:domain rdf:resource='#Person' />
    <rdfs:range rdf:resource='#Woman' />
</rdfs:ObjectProperty>

<Man rdf:ID='Adam' />
<Woman rdf:ID='Eve' />
<Man rdf:ID='Abel' >
    <father rdf:resource='#Adam'>
    <mother rdf:resource='#Eve'>
</Man>
<Man rdf:ID='Cain'>
    <father rdf:resource='#Adam'>
    <mother rdf:resource='#Eve'>
</Man>

```

There can be also defined a hierarchy of properties. In the running example, there can be defined a relation `parent-of` with domain and range `Human`. The `father` and `mother` properties can be then defined as subproperties with a more restricted domain or range.

Currently, RDFS is used as a language of RSS – Really Simple Syndication, a format for syndicating news and the content of news-like sites. According to the w3c web, the RDFS is now considered a part of RDF.⁶

The RDFS formalism is further used as a basis for further formalisms, so its formal definition is provided. The concepts and relations can be found in the RDF vocabulary (<http://www.w3.org/TR/2004/REC-rdf-schema-20040210/>). For definition purposes, supporting vocabulary was omitted (containers, collections, reification, and utility).

$$\Psi_{RDFS} = (\mathcal{C}_{RDFS}, \mathcal{R}_{RDFS}, \{\}, \{\}, \{\}), \quad (4.10)$$

where

$$\mathcal{C}_{RDFS} = \{rdfs:Resource, rdfs:Class, rdfs:Literal, rdfs:Datatype, rdf:XMLLiteral, rdf:Property\}$$

$$\mathcal{R}_{RDFS} = \{rdfs:range, rdfs:domain, rdf:type, rdfs:subClassOf, rdfs:subPropertyOf, rdfs:label, rdfs:comment\}$$

⁶source: <http://www.w3.org/TR/rdf-schema/>

4.7.6 DAML-ONT

The DARPA Agent Markup Language (DAML-ONT) was introduced in October 2000 as a joint work between US Department of Defence, industry and academia in both the US and the European Community. It is based on RDF Schema.

Again, its purpose was to become a standard in ontology language domain. After three months, the language was merged with OIL into DAML-OIL.

4.7.7 OIL

OIL (Ontology Interchange Language) is formalism developed in Europe at the same time as DAML-ONT. It has been mapped to RDFS in order to achieve maximal compatibility with existing applications [15].

OIL developers were aware about the RDFS reification problem. They decided to exclude reification from the new formalism.

The design of OIL expected three layers with increasing capabilities – Standard, Instance, and Heavy. Standard OIL supports the necessary mainstream modelling primitives with precise semantics, making complete inferencing viable. Instance OWL improves individual integration and included a database capability. Heavy OIL extends rule language and metaclass facilities.

During development of DAML-ONT and OIL, the research groups influenced each other and the languages were very similar.

4.7.8 DAML+OIL

Three months and one day after the announcement of DAML-ONT, joint EU/US committee released a new language, DAML+OIL. It combined both languages and provided both a model-theoretic and axiomatic semantics.

A part of the initial release was also a difference between DAML+OIL and the prior languages – there were 9 changes from DAML-ONT and 14 for OIL. The comparison⁷ shows, that OIL is more strictly defined while DAML-ONT is more RDFS compatible.⁸

Most of the publicly available ontologies is developed in DAML+OIL. The official site itself offers a library of ontologies: www.daml.org/ontologies.

For, DAML+OIL, the running example will not be shown, because it is identical to the one in the RDFS section. Only the `rdfs:Class` is replaced with `daml:Class` and the same is for `rdfs:ObjectProperty`.

Instead, a restriction is demonstrated:

```
<daml:Class rdf:ID="Person">
  <rdfs:subClassOf rdf:resource="#Animal"/>
<rdfs:subClassOf>
```

⁷source: <http://www.daml.org/2000/12/differences-oil.html>

⁸From the differences between OIL and DAML-OIL: “Arbitrary RDF cannot be used in OIL ontologies.” or “OIL has explicit ‘OIL’ instances; DAML+OIL relies on RDF for instances.”

```

    <daml:Restriction daml:cardinality="1">
      <daml:onProperty rdf:resource="#hasFather"/>
    </daml:Restriction>
  </rdfs:subClassOf>
</rdfs:subClassOf>
  <daml:Restriction>
    <daml:onProperty rdf:resource="#shoesize"/>
    <daml:minCardinality>1</daml:minCardinality>
  </daml:Restriction>
</rdfs:subClassOf>
</daml:Class>

```

In this example, a person is a subclass of animal and is required to have exactly one father. Furthermore, each person have at least one shoe size. The restriction defines an anonymous class (class of all things that have exactly one father). The person is then required to be a subclass of this anonymous class.

DAML+OIL allows to extend already existing classes, which simplifies usage of upper ontologies. It is possible to add required features to the used concepts without changing the imported file.

```

<daml:Class rdf:about="#Person">
  <rdfs:subClassOf>
    <daml:Restriction daml:maxCardinality="1">
      <daml:onProperty rdf:resource="#hasSpouse"/>
    </daml:Restriction>
  </rdfs:subClassOf>
</daml:Class>

```

4.7.9 OWL

Currently, OWL (Web Ontology Language) is the leading edge of formalisms designed for semantic web and a w3c recommendation for standard for ontology authoring. It started after two years of DAML+OIL development. A motivation was obviously clarification of its semantics in order to allow development of inferencing engines. A document describing changes between DAML+OIL and OWL⁹ consists of twelve points, in most cases only renaming or minor changes (e.g. OWL incorporates recent definitions of RDF and RDF Schema, including XML Schema data types).

The OWL language defines three sub-languages, similarly to OIL – OWL Full, DL (Description Logic), and Lite. The language and its flavours are precisely described in the w3c website: <http://www.w3.org/TR/owl-ref>.

The *OWL Full* is the OWL language without any restriction. Ontology in OWL Full can use any combination of OWL and RDFS constructs. OWL Full is a real superset of RDFS.

⁹source: <http://www.w3.org/TR/owl-ref#appD>

OWL DL is a maximal subset of OWL Full against which current research can assure that a decidable reasoning procedure is realizable. The most important restriction defined in OWL DL is a requirement to strictly separate classes, instances, properties, and types. Already OIL distinguished between classes and instances, but this feature has been lost in DAML+OIL. A second restriction of OWL DL is disallowed mix of OWL and RDFS constructs – only the OWL ones should be used. There are forbidden statements on RDFS resources and some other rules.

OWL Lite further restricts the constructs in order to provide a minimal useful subset, which can be easily implemented. Only simple class hierarchies can be built, there can be used property constraints and characterisations, and classes can be constructed only through intersection or property constraints.

In the hierarchy of Description Logics, the OWL DL is based on *SHOIN(D)* and OWL Lite on *SHIF(D)* logics.

The OWL formalism is currently the top of the developed formalisms for the semantic web. Therefore there will be provided a formal definition. The definition covers OWL Full, because the complexity makes no problem. Obviously, the other two flavours are in the \prec relation: OWL Lite \prec OWL DL \prec OWL Full. Furthermore, only OWL Full is compatible with RDFS, while the sub-languages deprecate the RDFS constructs.

The formal definition does not fully describes all concepts and relations available in OWL, because the list is rather long and can be easily obtained from the index of OWL elements (<http://www.w3.org/TR/owl-ref/#appA>). The definition 4.10 of RDFS is used.

$$\Psi_{OWL} = (\mathcal{C}_{OWL}, \mathcal{R}_{OWL}, \mathcal{S}_{OWL}, \{\}, \{\}), \quad (4.11)$$

where

$$\begin{aligned} \mathcal{C}_{OWL} &= \mathcal{C}_{RDFS} \cup \{owl:AllDifferent, owl:AnnotationProperty, \dots, \\ &\quad owl:TransitiveProperty\} \\ \mathcal{R}_{OWL} &= \mathcal{R}_{RDFS} \cup \{owl:allValuesFrom, owl:backwardCompatibleWith, \dots, \\ &\quad owl:versionInfo\} \\ \mathcal{S}_{OWL} &\text{ differs for OWL Full, DL, and Lite} \end{aligned}$$

The \mathcal{S}_{OWL} make the difference between the OWL flavours. For the OWL Full the set is empty and the more other two flavours add their restrictions, i.e. the set of OWL-DL restrictions is a subset of the OWL-Lite's one: $\|\mathcal{S}_{OWL-DL}\| \subset \|\mathcal{S}_{OWL-Lite}\|$.

The running example written in OWL (OWL Lite in this case) shows using the basic constructs:

```
<owl:Class rdf:ID = 'Person' />
<owl:Class rdf:ID = 'Man' >
  <rdfs:subClassOf rdf:resource = '#Person' />
</owl:Class>
```

```

<owl:Class rdf:ID = 'Woman' >
    <rdfs:subClassOf rdf:resource = '#Person' />
</owl:Class>
<owl:ObjectProperty rdf:ID = 'father'>
    <owl:label>father</owl:label>
    <rdfs:domain rdf:resource='#Person' />
    <rdfs:range rdf:resource='#Man' />
    <owl:minCardinality>1</owl:minCardinality>
    <owl:maxCardinality>1</owl:maxCardinality>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID = 'mother'>
    <owl:label>mother</owl:label>
    <rdfs:domain rdf:resource='#Person' />
    <rdfs:range rdf:resource='#Woman' />
    <owl:minCardinality>1</owl:minCardinality>
    <owl:maxCardinality>1</owl:maxCardinality>
</owl:ObjectProperty>

<Man rdf:ID='Adam' />
<Woman rdf:ID='Eve' />
<Man rdf:ID='Abel' >
    <father rdf:resource='#Adam'>
    <mother rdf:resource='#Eve'>
</Man>
<Man rdf:ID='Cain'>
    <father rdf:resource='#Adam'>
    <mother rdf:resource='#Eve'>
</Man>

```

An interesting extension of OWL is a project SWRL: A Semantic Web Rule Language,¹⁰ combining OWL (DL or Lite) and RuleML (Unary/Binary Datalog). SWRL enriches OWL by Horn-like rules.

Although OWL-Lite was designed to allow inferencing, even the most developed system for OWL processing, Jena2, does not provide this functionality. Its *OWL FB Reasoner* defines itself as “A useful but incomplete implementation of the OWL/Lite subset of the OWL/Full language.”¹¹

4.8 Non-ontology Formalisms

As an extension to the classic view on ontology were [considered] several “formalism” describing structures, but not commonly viewed as ontology. The reason for introduction

¹⁰source: <http://www.w3.org/Submission/SWRL/>

¹¹source: <http://jena.sourceforge.net/inference/index.html>

4 Ontologies

of these special cases is not the intention to mutual transformation, but

- convert common computer objects (files, source codes, database tables) to formalisms processable by tools of knowledge management, for example OWL, and
- allow set operations available in GOF (see section 5.6).

The first sample of such “ontology” is a directory tree. It holds a special kind of knowledge – how to keep order in a large amount of file with close meaning (source code, its documentation, tests, config files etc.). An example of public “ontology” is a definition of a feasible structure of a software project (appeared at <http://www.jroller.com/page/gru/20021206>, here is shortened version):

```
root\  
 .ejb\  
   .ejb-app1\  
      lib\  
      res\  
      src\  
      test\  
    res\  
      db\  
      scripts\  
  web\  
    web-app1\  
      lib\  
      res\  
      src\  
      view
```

(root of an ejb based application)
(dependent jars)
(resource files like ejb-jar.xml)
(src)
(Unittests)
(global resource files)
(root of all web applications)
(all servlet/Javabeen srcs)
(jsp/html pages)

A formal definition for the “filesystem formalism” denotes root as \top .

$$\Psi_{filesystem} = (\{directory, file, \top\}, \{SubDirectory/2, InDirectory\}, \{\top subDirectory directory, \top inDirectory file, directory subDirectory directory, directory inDirectory file\}, \{\}, \{\}) \quad (4.12)$$

A similar “formalism” are bookmarks stored in a web browser. Bookmarks can be viewed as filesystem, with substitutions bookmark \rightarrow file and folder of bookmarks \rightarrow directory.

Both directories and bookmark folders define structure of domain of interest, usually very simplified as the number of instances (files, bookmarks) is not large.

4.9 Software Support

In the former text numerous formalisms have been presented. Most of them have a formal background theory in some kind of logic (mainly first-order logic).

In practice, capabilities of ontology formalism are given by capabilities of software implementing the formalism. The software necessary to work with formalism require knowledge acquisition tools, query engines, graphical interfaces for representation of the content of knowledge bases etc.

Nowadays, there exist huge amount of available software suitable for particular tasks in developing semantic web.

In the following sections will be presented editors and libraries capable of handling ontology formalisms.

4.9.1 Editors

Ontology editors are employed as knowledge acquisition tools. They are usually restricted to one formalism and others are in a limited way by its I/O modules, i.e. through import and export.

Beside the presented editors, there are many others (OntoEdit, OIled etc.). The presented one are the most important in the context of this work. Both are based on frames, e.g. classical frame-based knowledge-base editors with support for classes, instances, slots, multiple inheritance, etc.

Apollo (CH)

ApolloCH (<http://krizik.felk.cvut.cz/apolloch>) is the main source of ontologies developed in the CIPHER project and is now being developed at the Czech Technical University in Prague. It is based on Apollo editor originated from The Open University, UK (<http://apollo.open.ac.uk/index.html>). The original core has been extended in several ways.

Unique features comprise comparison of two ontologies, copy data (classes, instances) between them, support for concepts not yet defined, fast walking through the knowledge base (move between concepts similarly to hypertext), etc.

The main purpose of ApolloCH is to prepare ontologies in the OCML formalism. Its internal format is XML for better cooperation with modern applications.

As it is a Java application, it makes possible to serve as a library for accessing Apollo files and as an engine for knowledge base manipulation. This feature is used in GOF framework (see 5.6).

The ApolloCH internal XML format is used as one of formalisms used in this thesis deals with. It replaces the OCML formalism, because it provides an interface and contains only the structural part of ontologies.

Protégé

It allows user to define a form for entering values of instances' slots. There is a evaluation core processing simple queries over the knowledge base.

Protégé is designed to support multiple formalisms [39]. Its extensible architecture allows development of customised knowledge-based applications.

Protégé provides so-called RDFS Storage Backend to work with RDFS-based formalisms (OWL, DAML), which meets some incompatibilities between the frames and RDF – Protégé does not support for example multi-class membership, i.e. a concept being an instance of multiple classes. On the other hand, the constraints on range of a property (slot) can be in Protégé a value of primitive type or multiple classes. A table comparing concepts used by Protégé and RDFS is at page 80.

Latest development of Protégé leads to support missing features in frames, for example subproperties. It can be done by defining a new facet describing, that the particular slot is a “subslot” of another one.

The Protégé homepage also offers a set of available ontologies in the format of this editor: http://protege.stanford.edu/plugins/type_ontologies.html.

4.9.2 Engines

An important piece of software are engines particular formalism and eventually evaluating various queries.

For accessing ApolloCH files the editor's files (JAR, Java Archives) were used as a library.

For processing RDFS and OWL, two libraries were examined: KAON and Jena. Besides reading and writing files in the mentioned formats, both libraries can store knowledge bases in memory or in a persistent base and provide inference support for some subset of DL.

None DL library provided simple and uniform interface for RDFS and OWL, but rather separated both formalisms and there was a technical complication. Although method for reading the Apollo files contains 10 occurrences of a method `kb.addConcept` (adding a concept to a knowledge base), the method using Jena library contains 52 occurrences of this method.

The Jena (again in version Jena2, www.hpl.hp.com/semweb/jena2.htm), open source project originating in the Hewlett-Packard Labs Semantic Web Research. Its base can be queried using the RDQL (RDF Query Language) standard. Jena is the most reliable platform for accessing RDFS/OWL knowledge bases.

The second library choosed was the KAON (KAON2 in these days, <http://kaon2.semanticweb.org/>) is a library with an API for programmatic management of OWL-DL ontologies and inference engine supporting the SHIQ(D) subset of OWL-DL. This includes all features of OWL-DL apart from nominals (also known as enumerated classes).

In the early stages were tested also OWLAPI as a simple interface to OWL files. It was able to read only several OWL files, so it was not further investigated. OGraph, a promissing library providing API to OWL, failed to process some of basic features of

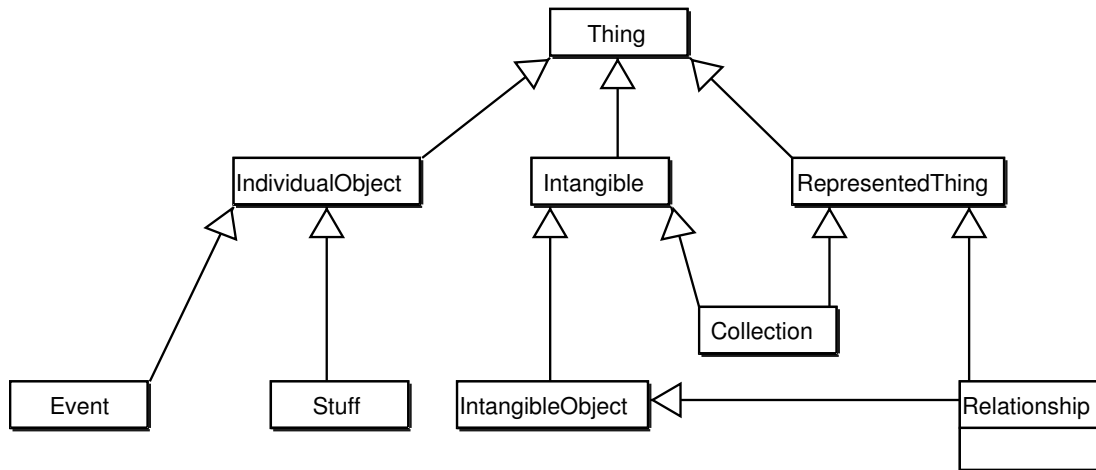


Figure 4.12: Upper concepts in Cyc

OWL, like subproperty with not fully defined range and domain. It also does not process `completelyFills`.

All the mentioned engines offer much more functionality than described here. The dark side of the advanced features is a requirement of a specific form of knowledge base. Although theoretically there is no problem, a practical experience shows incompatibilities between available ontologies and capabilities of the engines.

4.10 Upper Ontologies

Knowledge-based systems are expensive to build, test, and maintain. A huge amount of effort can be saved by using a standardised ontology. Available ontologies will cover exactly the application needs, but the existing ontologies can be used as a base covering the general concepts.

Upper ontologies, also known as top-level (Sowa, [45]) or general ontologies (Russel & Norvig, [44]), incorporate decisions about how to represent a broad selection of objects and relations. Domain ontologies refer to this upper level as a base of its common vocabulary. It allows different domain ontologies operate on a abstract level.

A design of an upper ontology differs between different projects, because there is no self-evident way of dividing the world up into concepts. The diversity is visible in figures 4.12–4.15 representing the top concepts in particular upper ontologies.

4.10.1 Cyc

The purpose and features of the Cyc upper ontology was already mentioned in section 3.4. It is the largest ontology in these days.

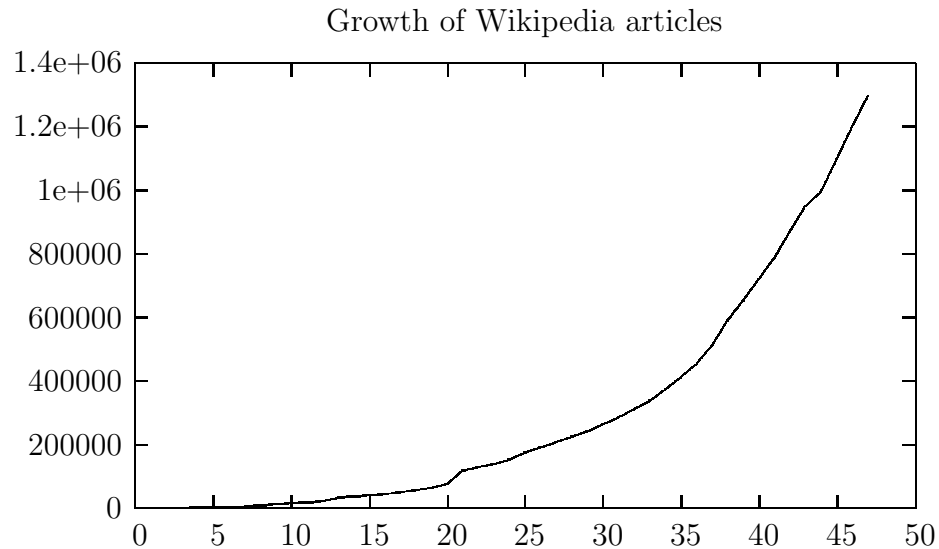


Figure 4.13: Wikipedia article count from January 2001 to December 2004

4.10.2 WordNet

Wordnet serves as an semantic lexicon of the English language. As its basic formalism it uses semantic network to represent synonyms, antonyms, hyponyms, and meronyms. It is developed from 1985. WordNet was developed by the Cognitive Science Laboratory at Princeton University under the direction of Professor George A. Miller (Principal Investigator) with many contributions by other people.

Currently (beginning of 2005) it contains more than 150.000 words.

4.10.3 Wikipedia

Another application of semantic networks is Wikipedia. Wikipedia is a Web-based encyclopedia. Unlike Cyc, it consists of a huge amount of articles to be read by people. The articles contain images and many hypertext links either to another articles or to external Internet sources. Wikipedia was used in this thesis as one of sources of information.

One of its goal is freedom – it is multi-lingual, “copylefted”, designed to be read and changed by anyone (even with the danger of vandalism). Thousand of users edit and maintain huge amount of articles about many concepts. Wiki stores also a complete history of articles, so it is possible to watch evolution of the text.

The software used for Wikipedia is an opensource program wiki, started by Ward Cunningham, and the whole database and web server is hosted and supported by the non-profit Wikipedia Foundation.

At the end of 2004 Wikipedia contained approximately 1.300.000 articles.

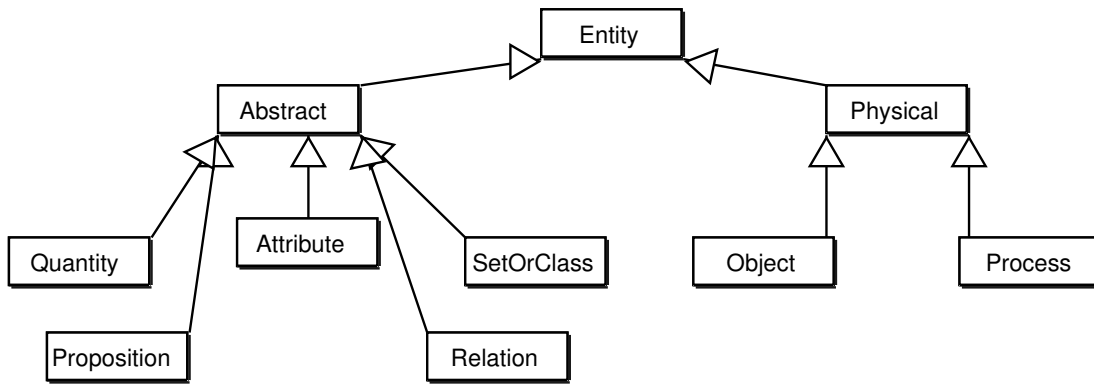


Figure 4.14: Upper concepts in SUMO

4.10.4 SUMO

The Suggested Upper Merged Ontology (SUMO, <http://suo.ieee.org/>) was one of intended subjects of transformation between formalisms.

SUMO is developed by the Standard Upper Ontology Working Group. It was created by merging a number of existing upper-level ontologies and provides a foundation for middle-level and domain ontologies. It has links to several languages – Hindi, Chinese, Italian, German, Czech, and English. Besides the core of the SUMO, there are available attached the Mid-Level Ontology (MILO) and ontologies of Communications, Countries and Regions, distributed computing, Economy, Finance, engineering components, Geography, Government, Military, North American Industrial Classification System, People, physical elements, Transnational Issues, Transportation, Viruses, World Airports.

As it is becoming an IEEE standard, it is important to involve the terms in SUMO into applications to accomplish the ideal of the semantic web – a common language for different applications.

SUMO concepts are mapped to the WordNet. Mapping SUMO to the WordNet is motivated especially to promote the use of SUMO in natural language understanding applications.

The SUMO consists of four basic parts. The first part, Structural Ontology, contains definitions for relations that serve as the framework for defining the ontology. The second part, Base Ontology, is similar to philosophical ontologies since it contains fundamental ontological notions such as abstract and physical entity and the distinction between objects and processes. It is basically a taxonomy containing the hierarchy of concepts with definition of each concept written as plain text. The third part consists of several sections. Each of these sections covers a domain that is to some degree standardised and the concepts of which are often referred to in various disciplines. Domains covered by this part of SUMO include Set/Class Theory, Graph Theory, basic arithmetics and units of measurement. There is also the Temporal section based on Allen's temporal relations and the Mereotopology section that contains a basic axiomatisation

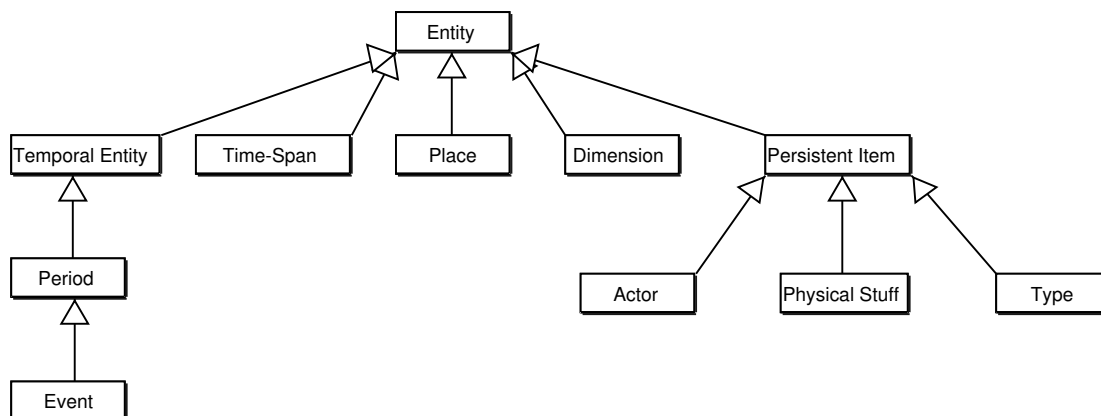


Figure 4.15: Upper concepts in CRM

of part/whole relations, as well as a normalisation of holes. The remaining part of the ontology consist of sub-hierarchies and axioms relating to process types (e.g. “Change-OfPossession” and “Touching”), object types (e.g. “Book” and “Fish”), and attribute types (e.g. “SocialRole” and properties of sensation).

Currently SUMO contains in all parts (including the domain ontologies) 20,000 terms and 60,000 axioms.

4.10.5 CRM

The CIDOC Conceptual Reference Model (CRM, <http://cidoc.ics.forth.gr/>) provides definitions and a formal structure for describing the implicit and explicit concepts and relationships used in cultural heritage documentation.

CRM is not upper ontology like Cyc or SUMO, it is on an edge between upper ontologies and so called *domain ontologies*. It defines top concepts, but concentrates more on topics connected to history.

The CRM ontology [22] is one of the upper ontologies widely used in culture heritage domain.

The CRM ontology has been used in the CIPHER project for modelling persons, places, events, and interrelations between them. This decision speeded up the annotation of all the 36 stories in the project. Also a cooperation of teams was simplified as both sides worked with a standard ontology and the ontology did not change during the work.

The CRM ontology suffers from not being a part of a bigger group of ontologies. During work on the CIPHER project was required a conceptualisation of family relations, not covered by CRM. These relations have been created in an application ontology (a common level defined within an application). It contradicts with ontology purpose – to consolidate used vocabulary.

4.11 Others

4.11.1 Dublin Core Metadata Initiative

The Dublin Core (DC) Metadata Initiative (<http://www.dublincore.org/>) is an open forum engaged in the development of interoperable online metadata standards that support a broad range of purposes and business models. DCMI's activities include consensus-driven working groups, global workshops, conferences, standards liaison, and educational efforts to promote widespread acceptance of metadata standards and practices.

A detailed investigation of the Dublin core shows, that it is not an ontology in our point of view, although it is often referred in context of ontologies. Dublin core does not define any structure. It defines only a set of attributes of resources. A good example is DC as a part of RDF:

```
1. <dissertation>
2.     <dc:author>Petr Aubrecht</dc:author>
3. </dissertation>
```

1. part of regular RDF in XML form, element denoting and object,
2. additional attribute defined by the Dublin Core.

Dublin Core can be described as a formalism with a grammar containing only relations

$$\Psi_{dc} = (\{\}, \{title, creator, subject, description \dots\}, \{\}, \{\}, \{\}). \quad (4.13)$$

A formalism with a DC extension is then a formalism with relations extended by the DC set

$$\Psi_X = (\mathcal{C}_X, \mathcal{R}_X \cup \{title, creator, subject, description \dots\}, \mathcal{S}_X, \mathcal{S}_X, \mathbb{A}_X). \quad (4.14)$$

5 Ontology Transformations Between Formalisms

5.1 Motivation

One of the main purposes of ontologies according to [17] is knowledge sharing and reuse. The whole previous chapter was dedicated to a description of some of the formalisms available. The formalisms have their specific advantages and drawbacks, they support particular features or focus on a simplification in some situation. Ontologies are created for a specific purpose in the most appropriate formalism satisfying needs of a narrow target community. The problem arises when the ontology is going to be used in another system supporting a different formalism. Although the main idea of ontologies is to consolidate hierarchy of concepts, the formalisms/formats used are mutually incompatible.

An unavailability of an upper ontology was one of the problems in the CIPHER project. Using upper ontologies saves huge amount of effort – it is not necessary to reinvent the wheel again and again and they provide a proven solution.

Unfortunately, the upper ontologies are developed in a different formalisms than OCML, which was used in the project. The two most popular options are unusable – SUMO is being created in a variant of KIF and for Cyc there has been developed a special language, CycL.

One of the outputs of the CIPHER project were several ontologies. As the project is concerned with history, there were developed ontologies of stories from the South Bohemia and the history of Bletchley Park in UK. The intention is to make the ontologies available to further processing and thus it is necessary to store it in a formalisms used by other research groups.

A reason for using OCML instead of OWL was the procedural capability of OCML. It was used in the ontology of time [28] and further in the Story Fountain portal. The portal serves as a prototype of the semantic web – the pages are described in ontologies and a search in the ontologies allows semantic questions and more advanced queries. More information can be found in [37].

For this purpose a framework has been prepared to translate these formalisms to and from the OCML. Later, the approach was made general and successfully tested as a transformation between multiple formalisms.

A more theoretical motivation for a generalised formalism can be the early architecture of Semantic web, proposing RDFS, Ontology vocabulary, and logic layers in figure 4.6 without clear interconnection between them. The approach presented in this text could be the missing interconnecting element. The RDFS standard is well known today with

reliable parsers, there already exist verified vocabularies (SUMO, Cyc) and processing logic operations can be done either in first-order logic (Prolog) or in CycL. All these parts can be connected by the presented framework.

The need of transformation of ontologies will be required in the near future. The demand will come from the semantic web domain, which plans to cover the whole world-wide-web, all its pages, applications, relations, etc. The idea is to use ontologies to declare meaning of the web page/application/service/what-so-ever. The service part of the world-wide web will be important, because it is a connection between internet and business and can be applied for example in b2b negotiation, electronic marketplaces, etc.

Although designers of ontology formalisms, especially developers from the W3C Consortium, declare, that their latest formalism is going to be the only formalism acceptable by the whole world, experience on the contrary shows, that particular formalisms have specific features, which make them feasible for particular tasks.

Researches have not yet reached consensus on which formalism is the most suitable (even for semantic web), which features or syntax is the most appropriate. It is likely that more formalisms emerge.

The w3c consortium pushes to use its own ontology standard – OWL (described in section 4.7.9). The standard itself shows, that the decision is more controversial; OWL itself exists in three versions with different expression power and so far there does not exist engine giving reliable results.

Lisp-based formalisms (like OCML, section 4.5.4) are bounded to a Lisp interpret and it is difficult to use such knowledge base from another programming language.

This diversity introduces incompatibilities between systems and requires transformation between different knowledge bases using different formalisms and different variants of storage (XML, 3-tuples, etc). Therefore the consolidating application has to handle both different storage formats, and various formalisms with mutually incompatible constructs.

As a result, there is a need to provide a way how to transform an existing formalism to another one. Typically there are at least two systems – one offering the data (source system) and one processing the data (target system, for example reasoner). In the further text a methodology for such transformation will be given.

5.1.1 CIPHER Project

The research project of the 5th Framework Programme of European Community entitled “Communities of Interest Promoting Heritage of European Regions” (CIPHER) – www.cipherweb.org – served as a practical testbed for the results of this thesis. Its main aim was to develop Cultural Heritage Forums, associated to regions, that empower communities to create sustainable online cultural content for themselves.

A group of tool for dynamic narrative authoring and presentation have been developed within CIPHER. An ontology editor *ApolloCH* have been developed by KMI and extended by CTU – there were added features like multilingual support, comparison of two ontologies etc. *Resource Annotation Tool and Outline Creation Tool* (RAT-O) allows

to create personalised annotations of any electronic resources using ontology concepts. *Dynamic Narrative Authoring Tool* (DNAT) is a tool for authoring conceptual graphs with ontology support. Having multilingual ontology, it can automatically translate labels of narrative annotations. *Temporal Inference Engine* processes temporal facts and answers queries.

An crucial part of the project is processing of ontologies. For the forum a set of ontologies annotating stories from the South Bohemia was prepared and presented. The work on annotations showed, that frame-based formalisms are very restrictive and it would be useful to allow ad-hoc relations with later finalisation. There were occasionally problems with an exact decision which usages of slots due to unclear semantics. Temporary relations would help faster annotations and allow later solution of such problems.

The image 5.1 shows a structure of collaboration between several CIPHER tools. Apollo(CH) is a source of ontologies, the PAT editor and RAT-O allows creation of annotation (targeted to spatial information or with a support for a story creation). The final stories are presented at the CH-Portal. Ontologies support all the activities, i.e. for semantic search.

In the beginning, there were a need of a ready-made ontology reuse. Upper ontologies were investigated, but no one was in a required formalisms. As a base was selected the CRM ontology (described in section 4.10.5), which was manually created in ApolloCH.

This was the moment a research allowing ontology migration between formalisms started.

5.2 Migration within One Formalism

A more frequently solved problem is sharing ontologies between system with one common formalism. Pieces of ontologies (usually data, instances) within one application migrate between two particular ontologies, i.e. information exchange between two knowledge bases.

This problem is often addressed by multiagent systems with independent agents (MAS) collecting knowledge and creating their own structured view of the surrounding world. At the moment two agents have to communicate, they have to consolidate their ontologies.

Also business to business (b2b) systems have to handle sharing data between two different ontologies and convert the data between them.

Nowadays, there exist several tools supporting (semi)automatic ontology sharing or conversion. They can be classified according to different features. The tools provide ontology merging, building of semantic bridges, and also reasoning done simultaneously on a number of ontologies.

An important aspect making the difference between the knowledge sharing tools is the use or necessity of instances. One type of tools are based on graph-matching techniques and use of synonyms to find the corresponding concepts and therefore they do not require instances. Another type of tools uses machine learning techniques either for classification of instances of one ontology using the classes of the other ontology or to build up a new

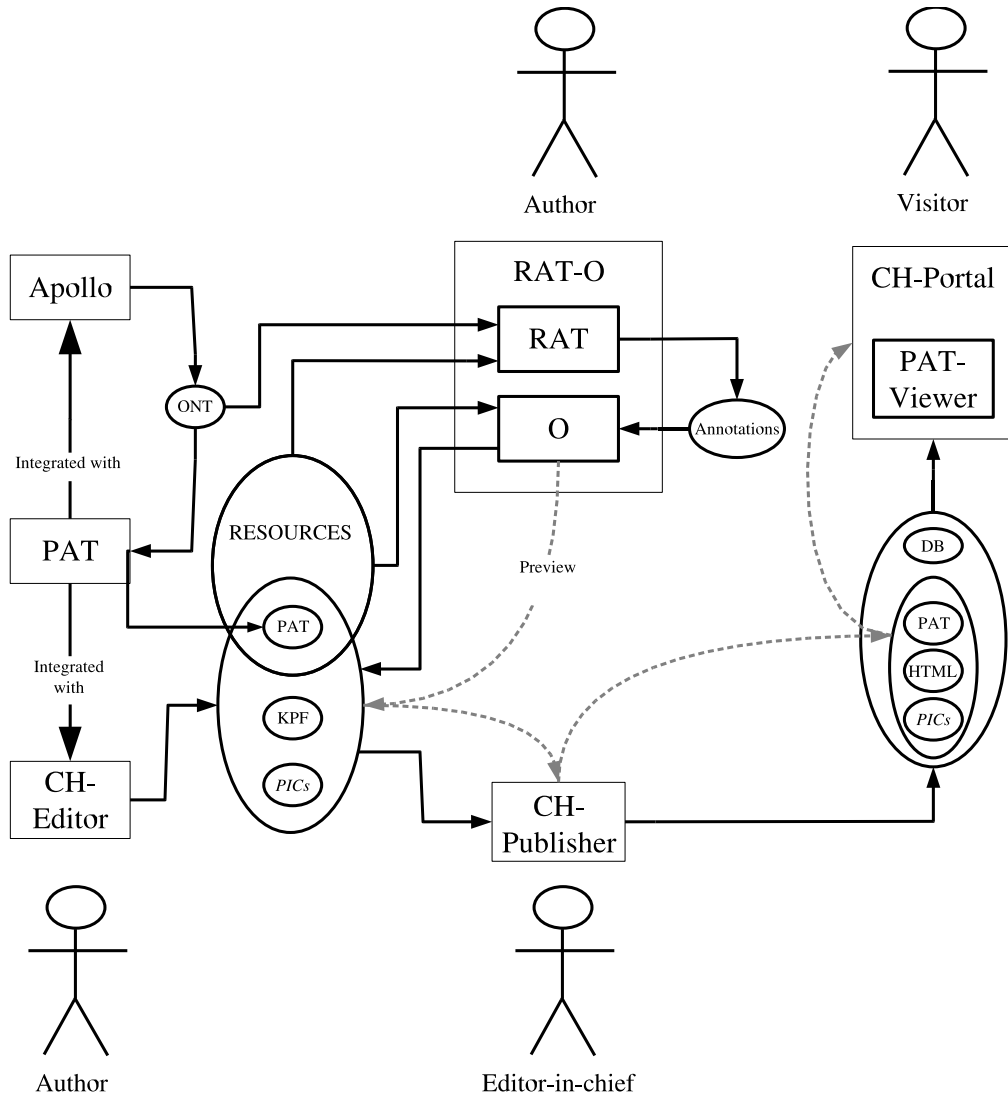


Figure 5.1: A schema of a part of CIPHER tools

ontology from the instance of all available ontologies. These tools require a relatively large number of different instances to work properly.

The tools for sharing knowledge within one formalism comprise projects:

Chimaera	http://www.ksl.stanford.edu/software/chimaera
OntoMerge	http://cs-www.cs.yale.edu/homes/dvm/daml/ontology-translation.html
FCA-Merge	http://www.aifb.uni-karlsruhe.de/WBS/gst/papers/2001/IJCAI01.pdf
ONION	http://dbpubs.stanford.edu:8090/pub/2000-20
GLUE	http://anhai.cs.uiuc.edu/home/papers/glue-handbook.pdf
PROMPT	http://www-smi.stanford.edu/pubs/SMI_Reports/SMI-2000-0831.pdf
Ontology Alignment	http://www.atl.external.lmco.com/projects/ontology

5.3 Known Approaches

Transformation of ontologies between formalisms is not very common. According to [49] there are three main approaches to transformation between different knowledge representation formalisms:

5.3.1 Mapping Approach

This approach leads to the lowest loss of information. A mapping is created which transforms expressions in the source formalism to expressions in the target formalism. Such mapping has to be defined for every pair of formalisms. Therefore it can be well adapted to the two specific formalisms. However the number of transformations that have to be designed increases sharply with the number of formalisms involved. It is also necessary to check properties of every transformation individually. That is why this approach is feasible only for systems working with a relatively small and fixed set of formalisms. An example of this approach is the OntoMorph system described in [8].

5.3.2 Pivot Approach

To avoid the necessity to create a large number of transformations one formalism is chosen as the pivot formalism. It has to be a formalism that is the most expressive of all the considered formalisms. For each of the other formalisms a mapping is designed that transforms expressions between the particular formalism and the pivot formalism. A transformation between two different formalisms is then done via the pivot formalism. The pivot formalism has to be very expressive to enable lossless transformation of all other formalisms into it. It has to be extended almost every time a new formalism is added to the system. Especially in case the system involves formalisms that are unlike each other e.g. formalisms based on description logic, formalisms based on frames, UML

etc., the pivot formalism would have to be quite complex. It is also difficult to design the pivot formalism so that it would not be biased towards one type of formalisms.

5.3.3 Layered Approach

The third approach uses a layered architecture containing languages with increasing expressiveness [49]. There has been an attempt by W3C to provide a standard group of languages that would be layered on top of each other, using RDFS as the layer. However other requirements on properties of the higher ontology languages, especially their decidability needed for reasoning, were more significant for their design than full backwards compatibility with the RDFS. The higher ontology languages such as DAML+OIL and OWL only use terms defined by RDFS as their basis. Except for OWL-Full the ontological languages do not cover RDFS completely. Some expressions valid in RDFS are not allowed in the other languages.

5.3.4 Family of Languages

In addition to the approaches described above a new approach called the Family of languages approach is proposed in [14]. It is a generalisation of layered and pivot language approach. In [14], the Family of Languages is defined to be:

Definition (*Family of Languages Property*) *A set of languages L_1, \dots, L_m satisfies the family of language property, iff they form a semi-lattice with respect to the coverage relation, i.e. if for every pair of languages L_i, L_j in $\{L_1, \dots, L_m\}$ there exists a language $L \in \{L_1, \dots, L_m\}$ such that $(L_i \prec L) \wedge (L_j \prec L)$.*

The coverage relation can be defined in a number of ways. It depends on the properties, which the transformation should preserve. There are four types of coverage relation: language-based coverage (one language is a subset of the other), interpretation-based coverage (there exists an interpretation-preserving transformation between the languages), consequence preserving and consistency-preserving. The last two of them imply a loss of information which is inherent in transformation from a more expressive language to a less expressive one.

5.3.5 Separated Worlds

Nowadays ontologies are developed in two basic areas – in multiagent systems, which tend to use Lisp-based formalisms (Ontolingua, Cyc, OCML) and semantic web, which prefers RDF/XML based formalisms (DAML, OWL).

Tools translating ontologies respect the boundaries and usually translate between near formalisms. There is almost no interaction between ontologies based on different base languages. Figure 5.2 shows few of the boundaries.

The goal of this thesis is to bridge the boundaries and allow to share ontologies between the worlds.

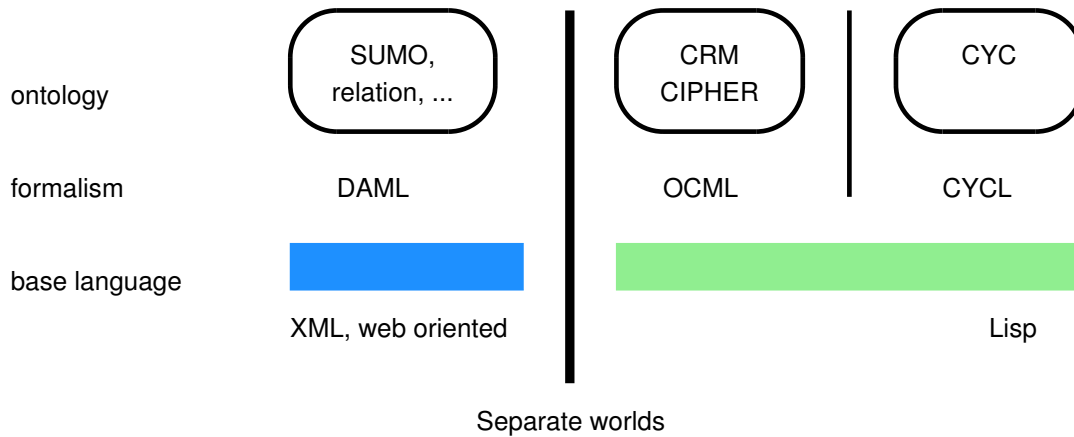


Figure 5.2: Separated world of ontology formalisms

5.4 Software Support

The most common way how to support multiple formalisms are I/O plugins for ontology editors. Almost every editor provides plugins for formalisms similar to the internal one and the information loss is increasing with difference between the internal and exported formalisms.

One of the leading ontology editors is Protégé, developed at Stanford Medical Informatics institute [47]. Protégé can be used to import, edit and save ontologies in CLIPS, RDF, DAML+OIL, XML, UML and OWL. For the first four formalisms is this functionality provided by I/O plugins. The OWL plugin has been added quite recently and it provides also modelling environment for development of OWL ontologies. The original frame-based metamodel of Protégé was extended to support constructs from description logic. The OWL Plugin provides a mapping between its API and Jena parsing library, which is used for importing and saving OWL ontologies. Transformation between metamodel of Protégé and external formats is lossless. However there is no mechanism for handling the situation of transformation between two external formats. For example, when an OWL ontology is imported to Protégé, exported to CLIPS via Protégé metamodel and then back to OWL again via Protégé metamodel, a lot of information is lost. E.g. in ontology `tambis-full.owl`¹ classes defined using `owl:EquivalentClass` are missing and information defined using `owl:Restriction` is lost.

5.4.1 Ontolingua

Ontolingua has been already described in section 4.5.3 in chapter about ontology formalisms. Here is important to emphasise, that Ontolingua has been designed as system for portable ontologies. Its *Frame Ontology* serves as a dictionary of available constructs.

¹<http://www.cs.man.ac.uk/~horrocks/OWL/Ontologies/tambis-full.owl>

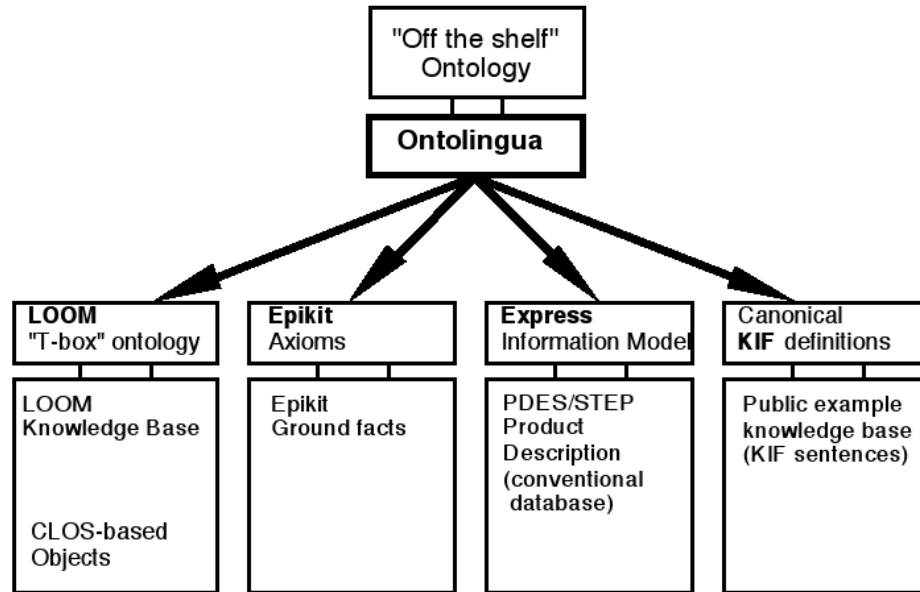


Figure 5.3: Ontolingua, translation from one formalism into multiple ones

When importing ontology from a particular formalism, the content is expressed by these constructs. The internal model is further translated to the target formalism. The subset of formalisms capabilities is given by the Frame Ontology.

5.4.2 Generic Frame Protocol

A project aiming to unify access to frame-based formalisms appeared in 1997 – Generic Frame Protocol (<http://www.ai.sri.com/~gfp/>) based upon the article P. D. Karp, K. Myers, and T. Gruber, "The generic frame protocol," in 1995. For the generic model of frame representation systems (with frames, classes, slots, etc.) it defines a set of access functions (e.g., get a frame by its name, change a slot's value in a frame). These functions can be used by an application to access knowledge stored in any compatible FRS (e.g., USC/ISI's Loom, SRI's SIPE-2 sort hierarchy, CMU's Theo, Stanford's Ontolingua). The implementation simply translates between the generic knowledge-base functions and an existing FRS-specific functional interface.

5.4.3 Open Knowledge Base Connectivity

A more general approach was chosen in the Open Knowledge Base Connectivity (OKBC, <http://www.ai.sri.com/~okbc/>) project. OKBC is a successor of GFP.

Similarly to GFP, OKBC is not a formalism (language) but only an interface to other formalism. It defines a basic set of function the underlying (frame-based) formalism have to be capable to answer. The set consists of constructs commonly found in frame

representation systems: classes, slots, constants, frames, facets, individuals and knowledge bases.

OKBC supports also tell&ask interface to the languages.

The last version of OKBC specification, version 2.0.3, was published in 1998.

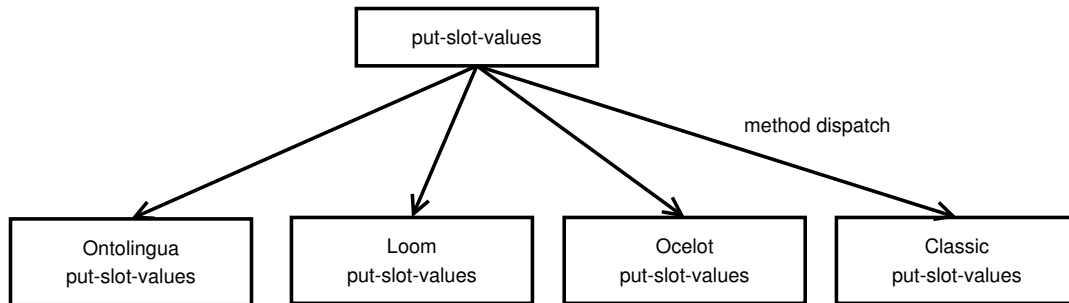


Figure 5.4: The architecture of the Common Lisp implementation of OKBC.

5.4.4 Chimaera

Chimaera (<http://www.ksl.stanford.edu/software/chimaera>) is a powerful system that supports manual merging of multiple ontologies together and diagnosing individual or multiple ontologies.

Chimaera supports multiple formalism on both input and output. In fact, it operates through OKBC interface.

The ontologies are compared and merged. Chimaera resolves in the first step class-name conflicts (decompose conflicting classes) and in the second step handle slots of the classes modified in the prior step. Chimaera offers several methods, how the issues can be solved. The software follows a wide range of available solutions of any possible problem.

After the merging part, ontology is aligned. The software concentrates on taxonomy fixing; it defines rules, which cannot be broken (e.g. a cycle in subclass hierarchy).

Ontology can also be aligned without the merging part.

Although Chimaera can communicate multiple formalisms, its internal model is based on a subset of OKBC and thus only intersection of the subset and input and output formalisms is migrated.

5.4.5 Further Tools

Ontomorph	http://www.isi.edu/~hans/ontomorph/presentation/siframes.html
java2owl	http://www.daml.org/2003/10/java2owl/
excel2rdf	http://www.mindswap.org/~rreck/excel2rdf.shtml
SWRL	http://www.daml.org/2004/05/swrl-translation/Overview.html

5.5 Formal Definitions

In this section a formal definitions of transformation between ontologies will be given. There will be also investigated a possibility of definition of a lossless transformation.

Definition 5.1 (Ontology Transformation Between Formalisms). *Ontology transformation between two formalisms is a function mapping ontology Ω_s in formalism \mathcal{F}_s to ontology Ω_t in formalism \mathcal{F}_t .*

$$\tau_{s,t} : \Omega_s \rightarrow \Omega_t$$

More practically, a transformation between ontology grammars is specified. Again, only structural part of the grammar is transformed ($\mathcal{C}_{\mathcal{F}}$, $\mathcal{R}_{\mathcal{F}}$, and $\mathcal{S}_{\mathcal{F}}$).

Definition 5.2 (Ontology Transformation Between Ontology Grammars). *Transformation between ontology grammars is a function mapping ontology grammar Ψ_s in formalism to grammar Ψ_t .*

$$\tau_{s,t} : \Psi_s \rightarrow \Psi_t$$

An important question is the amount of information lost during such transformations. Unfortunately, it can be shown, that due to incompatibilities between the existing formalisms it is impossible to guarantee that the transformation will be information lossless.

For example, when transforming between the OWL (section 4.7.9) and taxonomy formalisms, properties defined in OWL must be completely omitted. It means the \mathcal{C} set is smaller ($\|\mathcal{C}_{OWL}\| > \|\mathcal{C}_{taxonomy}\|$). On the other hand, some features must be approximated, e.g. restrictions. When the information about restrictions on particular class disappear, it can be replaced by adding relations *subclassOf* and thus enlarging of the \mathcal{R} set ($\|\mathcal{R}_{OWL}\| < \|\mathcal{R}_{taxonomy}\|$).

The only possibility how to compare results of such transformation, is when both ontologies are in the same formalism, i.e. only after carrying out the double-transformation (from \mathcal{F}_s to \mathcal{F}_t and back):

$$\tau_{t,s}(\tau_{s,t}(\Omega_s)) \stackrel{\subseteq}{=} \Omega_s. \quad (5.1)$$

Definition 5.3 (Lossless transformation of ontology between formalisms). *If both $\tau_{t,s}$ and $\tau_{s,t}$ transformations are lossless, $\tau_{t,s}(\tau_{s,t}(\Omega_s)) = \Omega_s$.*

Theorem 5.1 (Lossless transformation can be defined between formalisms in \prec relation). *If there are two formalisms \mathcal{F}_s and \mathcal{F}_t in \prec relation, there can be defined a lossless transformation from \mathcal{F}_s to \mathcal{F}_t :*

$$\Psi_s \prec \Psi_t \Rightarrow \tau_{t,s}(\tau_{s,t}(\Omega_s)) = \Omega_s$$

A proof of this theorem is obvious as the formalism \mathcal{F}_t contains all constructs available in the formalism \mathcal{F}_s , see the definition 4.6 at page 24.

A lossy transformation can both loose information (some concepts are missing in the double-transformed ontology), but there can be also more concepts. Reasons were shown earlier in this section.

Definition 5.4 (Purely lossy transformation). *A purely lossy transformation does not approximate constructs in the source formalism by construct of the target one, but omits all constructs without exact counterpart.*

Theorem 5.2 (Purely lossy transformation keep the \prec relation). *If there are two purely lossy transformations $\tau_{s,t}$ and $\tau_{t,s}$, they keep the \prec relation between the ontologies:*

$$\tau_{s,t} \text{ and } \tau_{t,s} \text{ are purely lossy} \Rightarrow \tau_{t,s}(\tau_{s,t}(\Omega_s)) \prec \Omega_s$$

Again, a proof is obvious from the definitions 4.6 and 5.4.

5.6 Generalised Ontology Formalism

Babelfish: A living fish which, when placed in your ear, will live there and translate any form of language for you.

(*Douglas Adams, The Hitchhiker's Guide to the Galaxy*)

The approach presented in this thesis is a syntactical transformation of ontologies using an internal model, consisting of concepts and binary relations between them. The most important feature is a tendency to define an abstract formalism with only few symbols. The formalism describing the model is called Generalised Ontology Formalism (GOF).²

Other attempts to solve ontology migration between multiple formalisms tend to using rich languages covering all features of desired formalisms. Such language become obsolete, whenever a new formalism with a new construct appears.

On the contrary, GOF lowers the limitations of the formalism in order to be able to describe a wide range of possible situations, even not yet investigated. This concept does not require extension of the model language for every new supported formalism. An example of a graph representing a knowledge base in GOF is in figure 5.10, page 71.

The basic idea of GOF is similar to RDF, but the binary relations are not concepts themselves and have no attributes. This allows drawing relations as arrows and the knowledge base as a graph. RDF based formalisms are hard to be drawn – arrows (properties) can be subclasses of other arrows. The usage of namespaces in GOF have been inspired also by RDF.

Naturally, the best results can be achieved with a design of a transformation between two particular formalisms directly, with maximum attention to the least detail comparing to introduction of a general transportation formalism (principle described in section 5.3.1). But regarding the number of available formalisms and the expectation that there will emerge soon new ones, the direct mapping between all formalism is unfeasible. At the moment when a “winner” formalism emerge, there can be precisely specified transformations between all the formalisms and the final one. Until then, writing specialised transformations for provisional formalisms, which could be soon abandoned, is wasting of time.

The methodology presented in the further text defines six relations, which can appear in the knowledge base. The selected relations represent the basic relations between concepts found in all considered formalisms and express the structural part of ontologies. The design is focused mainly on frame-based formalisms and description logic, but it can also be used for other systems like lattices, topic maps, or even ER diagrams, Java class hierarchy, etc.

To access different formalisms, there are so-called gates. The gates read or write a corresponding formalism and transforms ontologies in the formalism into the internal form. Gates contain also a metamodel of the formalism (formalism-specific ontologies).

²During the work on GOF it had a code name BabelFish, because it should serve as a connector between various (ontology) languages, similarly to the fish in *The Hitchhiker's Guide to the Galaxy*.

To achieve more accurate transformation results, it is possible to define a mapping between the gate ontologies for selected pairs of formalisms. Such transformation then takes advantage of knowledge, which concept in the source formalism corresponds to which concept in the target formalism.

The approach chosen in this thesis is similar to the *Pivot Approach* presented in section 5.3.2. Ontologies are translated into internal formalism and then to the target one. Instead of using the richest language, a simple one is selected. The *Mapping Approach* can be easily achieved by providing the mapping between gate ontologies. And the idea of the *Layered Approach* and the *Family of Languages* is in general unusable, because the languages considered do not form a lattice, i.e. there is no (or small) intersection among the languages.

The methodology has been used in the CIPHER research project for re-using upper ontologies (like SUMO or CRM, sections 4.10.4 and 4.10.5) from formalisms not covered by the CIPHER tools (DAML, OWL).

It is further shown that an extension of the SumatraTT system ([2]) can be used to implement such transformations in a graphical user interface. All the gates of GOF and all supported operations are represented by input/output modules in SumatraTT. The graphical interface is used for easier setup of the transformation and for testing. Details of SumatraTT are described in the chapter 6.

5.6.1 Evolution of Relations

The original idea started with only two kinds of relations in the beginning: *is-a* and *has-a*. The relation *is-a* should cover both the relations *subclassOf*, *instanceOf*. The relation *has-a* is the selected mapping for properties definitions and assignment values to properties of instances.

The running example in the intended formalism is in figure 5.5. For simple formalisms (e.g. taxonomies) it would be sufficient, but use of properties introduces ambiguity.

Moreover, there is a strong difference between *subclassOf* and *instanceOf* and between property definition and assignment in almost all the studied formalisms. For this purpose, the original set of two relations has been extended to:

- relations describing the structure – *subclassOf* (\rightarrow) and *instanceOf* (\rightarrow),
- relations defining properties *hasDomain* (\leftarrow) and *hasRange*
- relations for assignment values to particular properties of a concept *hasValue* (\rightarrow) and *propertyOf* (\leftarrow)

This separation of definitions allows the system to distinguish more precisely between terms of the studied formalisms. The extension to six relations allows to have unambiguous graph. Figure 5.10 shows the running example using all six relations.

In the next section the model will be described thoroughly.

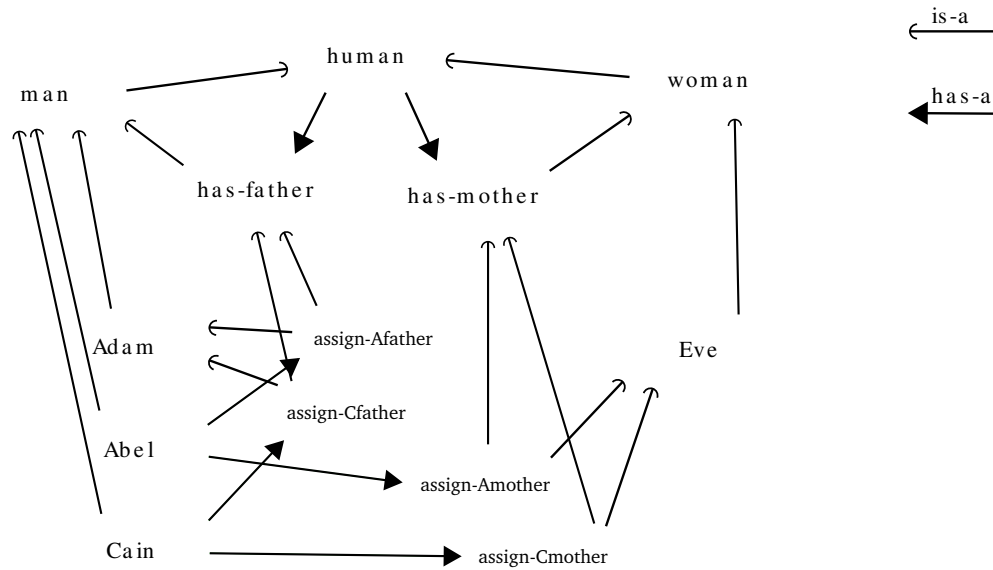


Figure 5.5: Running example using *is-a* and *has-a* relations

5.6.2 Generalised Ontology Formalism Definition

The Generalised Ontology Formalism can be most easily described as a graph with a set of uniform vertices (concepts) and six kinds of edges (relations). The edges represent different relations between the concepts.

The set of relations consists of:

instanceOf (\leftarrow) is used for decreasing the abstractness of the concept. It corresponds to the *is-a* relation in frames.

subclassOf (\rightarrow) expresses the specialisation relation between a more general and a more specific concepts.

has-domain (\leftarrow) “domain of a property”

has-range (\rightarrow) “range of a property”

propertyOf (\leftarrow) “an assignment of a value to an instance of a property domain”

has-value (\rightarrow) “a particular value of a property”

For better view into the intended use of relations, their meaning in frame-based formalisms is given in the following list:

instanceOf, \leftarrow – a relation between a class and its instance ($Adam \leftarrow man$) and also a relation between slot definition and an assignment ($assign-Afather \leftarrow has-father$).

subclassOf, \rightarrow – inheritance ($man \rightarrow human$)

has-domain, \leftarrow – $has-father \leftarrow human$ means a class $human$ has a slot $has-father$.

has-range, \square – $has-father \square man$ means, that a type a slot $has-father$ is a class man .

propertyOf, \ll – $assign-Afather \ll Abel$, the assignment concept must be an instance of a property, which domain is a class; the other side of \ll must be an instance of this class: $\beta \ll \mathcal{J}_A \Rightarrow \exists \alpha, \beta \rightarrow \alpha \wedge \exists \mathcal{A}, \alpha \leftarrow \mathcal{A} \wedge \mathcal{J}_A \rightarrow \mathcal{A}$.

has-value, \rightarrow – $assign-Afather \rightarrow Adam$, the assignment concept must be an instance of a property, which range is a class; the other side of \rightarrow must be instance of this class: $\beta \rightarrow \mathcal{J}_B \Rightarrow \exists \alpha, \beta \rightarrow \alpha \wedge \exists \mathcal{B}, \alpha \square \mathcal{B} \wedge \mathcal{J}_B \rightarrow \mathcal{B}$.

Now, GOF ontology grammar can be defined in the way introduced in chapter 4:

$$\Phi_{GOF} = (\{concept\}, \{-\circ, \rightarrow, \leftarrow, \square, \ll, \rightarrow\}, \{\}, \{\}, \{\}) \quad (5.2)$$

In the further text there will be used infix notation of GOF relations, i.e. “ $\rightarrow (A, B)$ ” will be written as $A \rightarrow B$.

The following several figures show basic shapes expressing common constructs in the analysed formalisms.

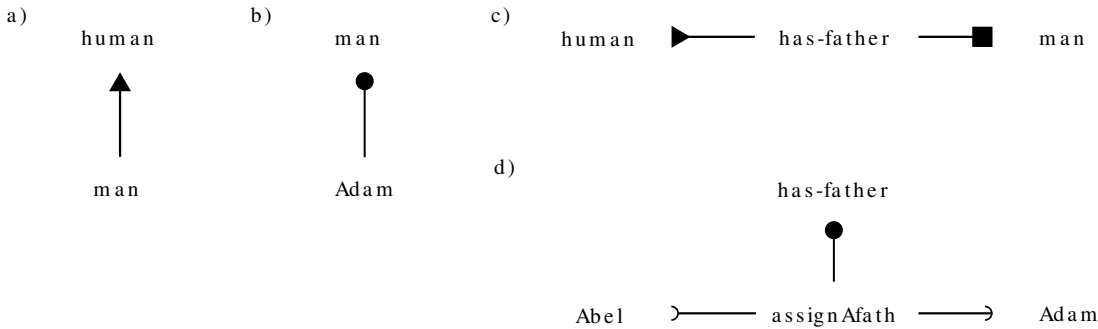


Figure 5.6: Examples of GOF relations

The most important relation is a subsumption. Figure 5.6a shows this relation. Very similar is making instance from a class, showed in figure 5.6b. The instance relations is also used for properties and assignments.

A definition of a property/slot requires multiple concepts and relations. A property is represented as a concept with two relations – range and domain. This simplest situation is in figure 5.6c. A use of property (in this work called assignment) is in figure 5.6d. The assignment concept relates to the definition property concept (using instanceOf relation), the source instance (using valueOf) and the value of the assignment (using hasValue).

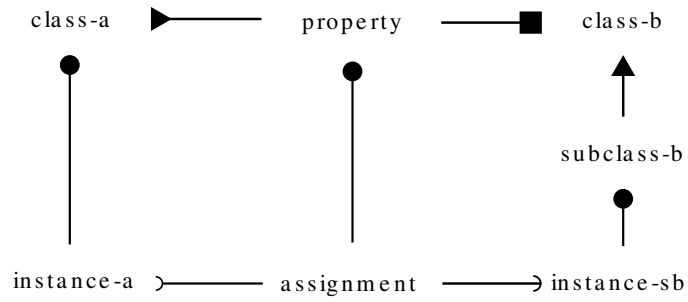


Figure 5.7: Sample Ontology in GOF

A simple ontology with all relations is in figure 5.7.

GOF formalism defines no restriction how the relations have to be combined. It allows to cover wide range of possible situations.

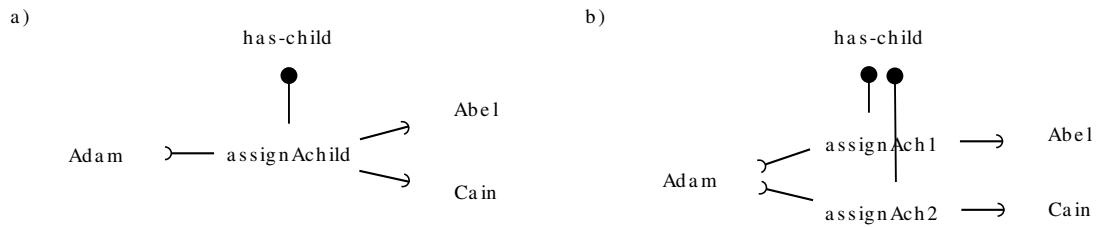


Figure 5.8: Variants of arrays in properties

For example, assigning a vector value into a slot in frames can be done by multiple \rightarrow relations (5.8a), similarly multiple values can be done by multiple \leftarrow relations (see figure 5.8b).

Description logics allow to define properties without determination of domain or range. Such case is shown in figure 5.9. In frame-based formalisms, this situation can be solved either by using a \top concept or omit such property at all.

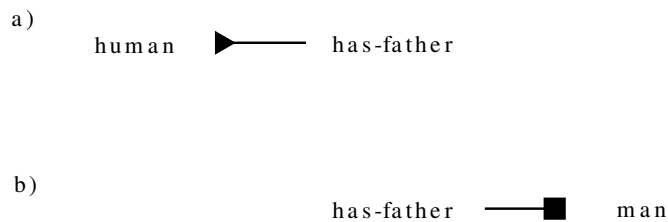


Figure 5.9: Partially defined properties in DLs

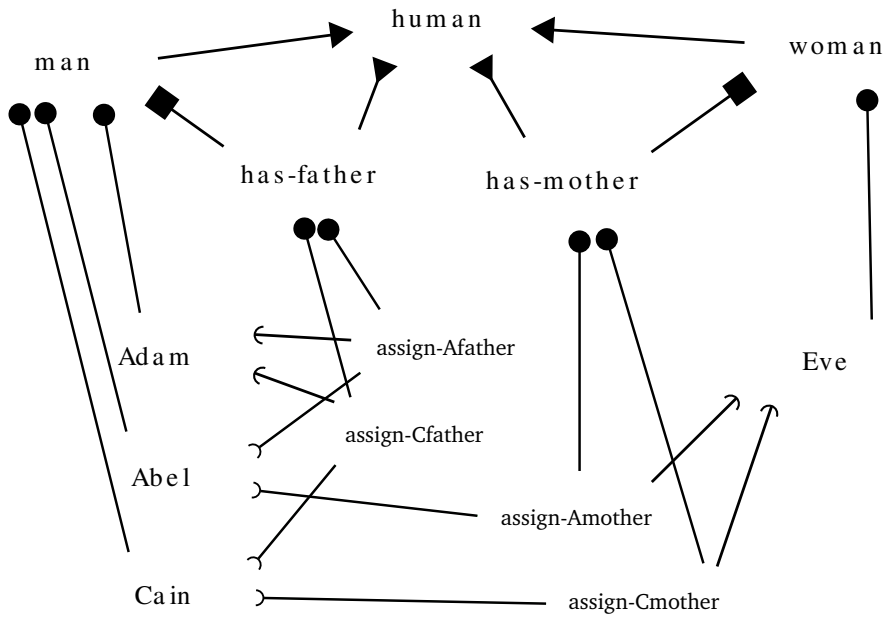


Figure 5.10: Running example using the generalised ontology formalism

To simplify merging of concepts from different sources we use also a namespace as a part of the literal in the same way XML does. Finally, the namespace and a name of a *concept* becomes an identification of the concept.

At the end of this section, a whole figure of the running example is given in figure 5.10.

5.6.3 Gates

A part of the whole generalised ontology formalism design is an idea of gates to particular formalisms. It is similar to I/O modules of ontology editors. Their role is to transform information from various formalisms into the internal GOF formalism. A schema with gates is in figure 5.11.

5.6.4 Formalism-Specific Ontology

When a gate is going to move ontology from GOF to its native formalism, it has to map concepts in the internal form into concepts used in the output formalism. For this purpose, ontology specific to the formalism can be specified. Similar definitions exist for all RDFS based formalisms mentioned in this thesis.

The formalism-specific ontology (FSO) is ontology of the formalism, which contains terms like Class, Subproperty, Restriction, etc. and expresses relations between them. In the further text there will be shown FSOs for the most important formalisms. They have been already presented in [3].

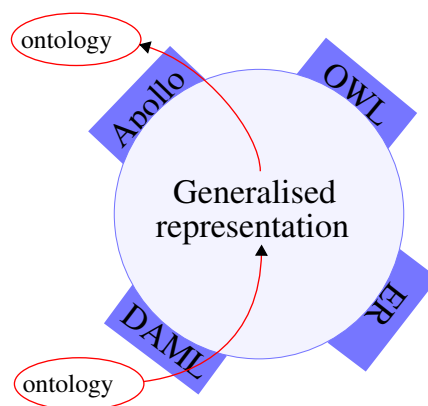


Figure 5.11: GOF Gates



Figure 5.12: Piece of ontology to be mapped

GOF framework provides a tool for mapping ontology's internal form to FSO, called *mapping engine*. The *mapping engine* is further described in the following section.

5.6.5 Mapping Engine

The mapping engine facilitates gates to map ontology concepts in GOF formalism to concepts of a native formalism (FSO), i.e. to decide, which concepts are classes, instance, properties etc. based on patterns recognised by the gate. The patterns are expressed in the $\mathcal{S}_{\mathcal{F}}$ set of the formalism grammar, see examples in section 4.3.3.

Gates do not have to use the engine, but it significantly simplifies its programming.

For example, frames's relation *subclassOf* (\rightarrow) is defined only between classes, e.g. the engine rule has form $con_{class} \rightarrow con_{class}$. Description logic can have also subproperties, so there are two rules – the one already shown and $con_{property} \rightarrow con_{property}$.

An advantage of the default engine is its ability to minimise information loss if GOF form is invalid with respect to the set of rules. It identifies the smallest set of invalid relations, which are then ignored or manually fixed.

Unfortunately, it can be proved that the mapping problem is equivalent to the graph colouring problem and thus non-polynomial. Fortunately, the rules are very restrictive in our case and thus the variable part is small.

Complexity of the Mapping Problem

The mapping engine uses a set of letters to express different roles of concepts in the native formalism. The designer of the particular gate designs a set of rules, how the

I	→	C
A	→	S
A	≠	I
A	≠	C

Table 5.1: Example of mapping rules

letters can be combined according to the used GOF relations. For example, let us take a small piece of GOF ontology shown in figure 5.12 and rules enumerated in table 5.1. Then the engine has to evaluate the expression

$$\left(((1 = C) \wedge (2 = I)) \vee ((1 = S) \wedge (2 = A)) \right) \wedge \left(((2 = I) \wedge (3 = A)) \vee ((2 = C) \wedge (3 = A)) \right). \quad (5.3)$$

The solution is

$$\begin{aligned} 1 &= C \\ 2 &= I \\ 3 &= A. \end{aligned} \quad (5.4)$$

Definition 5.5 (Basic Mapping Problem). *Let us have a set of letters \mathcal{L} and a set of rules in the form $X \rightarrow Y$, where X and Y are from \mathcal{L} , \rightarrow is one of the relations defined in GOF formalism, and ontology is in GOF formalism.*

The task is to assign one letter from \mathcal{L} to each concept in the ontology and satisfy all the rules.

Theorem 5.3 (Mapping Problem Complexity). *The mapping problem is a non-polynomial problem for $\|\mathcal{L}\| \geq 3$.*

Proof. The graph colouring problem is in the NP-complete class for number of colours greater or equal to three [12] and can be transformed to the mapping problem.

Transformation of the colouring problem to the mapping problem: the colours are mapped to letters and the set of rules will be the set $\{C_1 \rightarrow C_2 \mid C_1, C_2 \in \mathcal{L}, C_1 \neq C_2\}$. \square

The mapping engine solves a slightly more difficult problem, because if there exists no valid mapping, it tries to find the best mapping while ignoring as few relations as possible. The algorithm used for searching is based on approach known as *iterative deepening*.

5.6.6 Models of Selected Formalisms

OCML, Frames

Frame based formalisms have very simple schema. It consists of classes, which contain slots. An instance can be derived from a class. Values can be assigned to slots either in

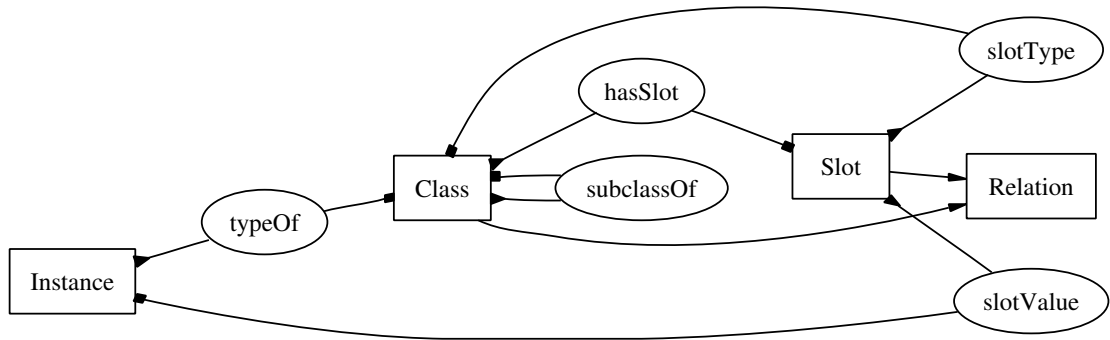


Figure 5.13: FSO of frames

C	\rightarrow	C	inheritance between classes
I	\rightarrow	C	instance of class
A	\rightarrow	S	assignment is an instance of a slot definition
S	\rightarrow	C	range of a slot is a class
S	\rightarrow	C	domain of a slot is a class
A	\rightarrow	I	assignment is connected to a particular instance
A	\rightarrow	C	... or a class (default value)
A	\rightarrow	I	a value of assignment is either and instance (including literal)
A	\rightarrow	C	... or a class (ApolloCH extension)

Table 5.2: Mapping rules for frames

class definitions (default values) and/or instance definitions (more common way). Values can be either literals or instances. The schema of frames is shown in figure 5.13.

As a small extension, ApolloCH also allows assigning also a class to a slot (fulfilling a request from practice). The figure can be very simply modified to express this extension – it is sufficient to add a \rightarrow relation between *slotValue* and *Class*.

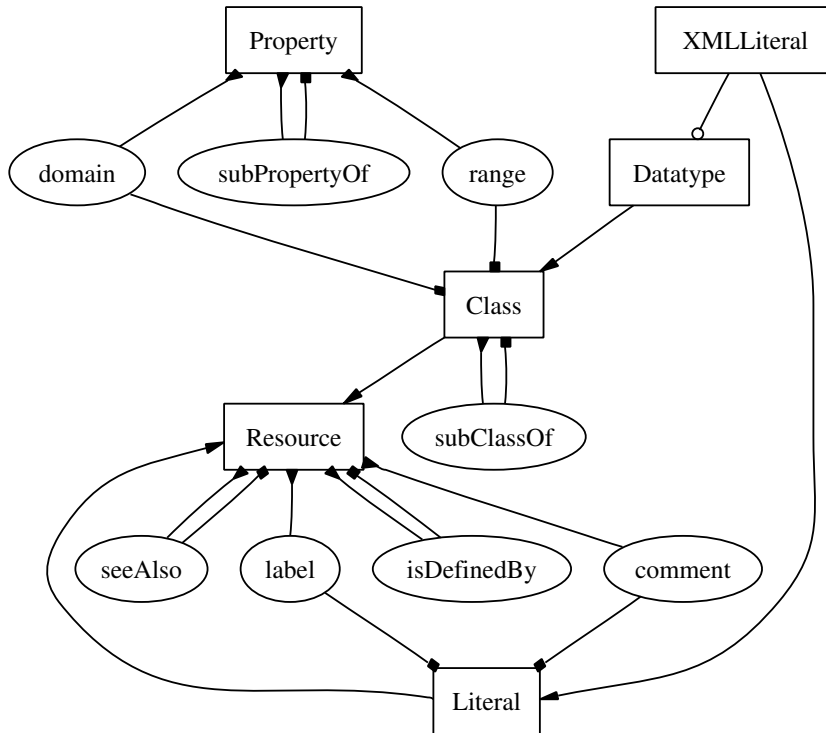
OCML have much more features, but all of them are procedural and thus not displayed in the figure.

Also a set of rules for the mapping engine is rather simple and is listed in table 5.2. In this table, C is a shortcut for *Class*, I – instance, S – definition of a slot, A – assignment, i.e. particular value of a slot.

RDFS

The RDFS FSO is slightly larger, because it contains explicitly defined all kinds of relations (domain, range) and it includes features not present in frames – *subPropertyOf*, *isDefinedBy*, *comment* etc. The model is based on the official w3c model of RDFS standard expressed in GOF.

Spurious relations are connected to the *XMLLiteral* concept. The *XMLLiteral* is a



Note: In order to simplify the picture, instances of Class are represented as rectangles and instances of Property as ovals. This notation removes plenty of \rightarrow edges.

Figure 5.14: FSO of RDFS

subclass of Literal and an instance of Datatype at the same time. The more generic concept Literal is an instance of Class.

OWL

Also the OWL FSO is the official w3c model expressed in GOF form. It is obvious from figure 5.15, that the OWL formalism definition is rather complex.

Moreover, two parts are missing in the figure – collections and restrictions are not mentioned for two reasons. First, these two parts are even more complex than this base. Second, the structure of collections and restrictions is not as important for GOF.

A part of OWL is also RDFS, which is omitted, too. Thus, the complete figure of the OWL FSO would be very complicated.

The mapping rules for the OWL formalism are listed in table 5.3.

5.6.7 (Un)Informed Transformation

The formalism-specific ontologies can be used also in transformations.

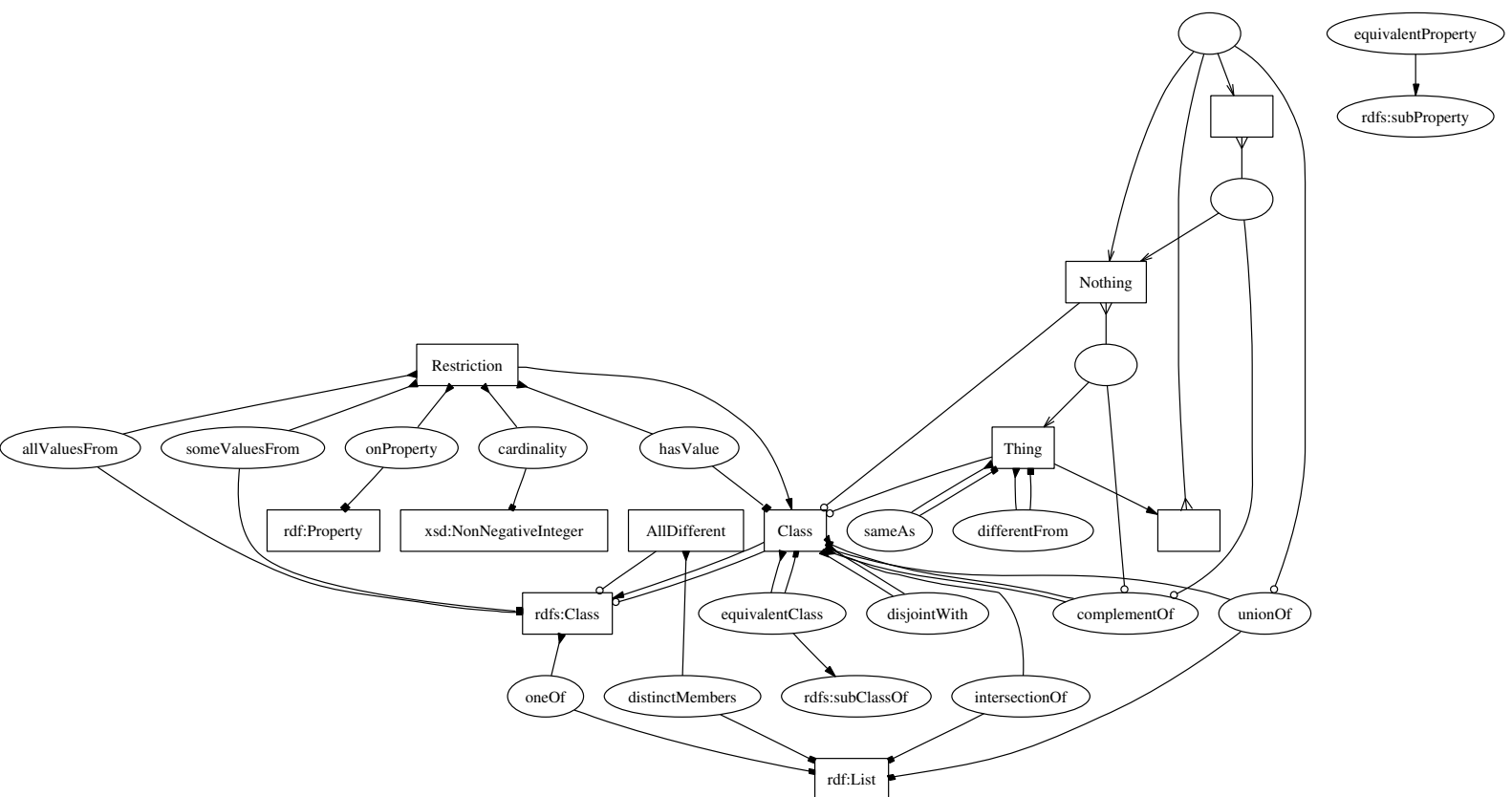


Figure 5.15: FSO of OWL

C	\rightarrow	C	inheritance between classes
P	\rightarrow	P	subproperty
P	\rightarrow	C	property range
P	\leftarrow	C	property domain
P	\rightarrow	P	a range of a property can be another property
A	\leftarrow	I	value of property of instance
A	\leftarrow	C	value of property of class
A	\rightarrow	I	value of property can be an instance...
A	\rightarrow	C	...or a class
P	\rightarrow	I	default value is an instance...
P	\rightarrow	C	...or a class
I	\circ	C	instance is an instance of a class
A	\circ	P	assignment (value of property) is an instance of property

Table 5.3: Mapping rules for OWL

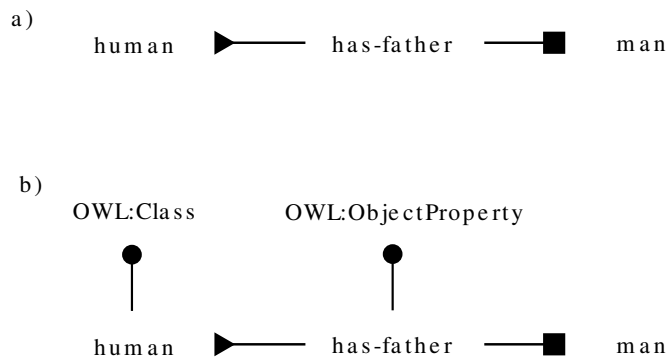


Figure 5.16: Ontology in GOF without (a) and with FSO (b)

If the transformation provided by a gate is lossless (the information can be completely expressed by the constructs of GOF), GOF can be seen as an ideal medium.

This is not the case of more complex formalisms. Some features (like some kinds of restrictions) can be simulated by a specific structures. After converting these structures back, they are interpreted in a different way and result in a different ontology.

There is an obvious demand to keep all information when converting from one formalism to GOF and back. For this purpose a special case has been defined and called *informed transformation*.

The gate can connect concepts of the processed ontology to the concepts in its FSO. An example of difference between ontology in GOF with and without FSO is in figure 5.16.

There then can exist a special transformation between two given formalisms using a connection between ontology in GOF and a FSO. This transformation takes advantage of knowledge of the source and target formalisms and in-advance-specified relations between corresponding concepts.

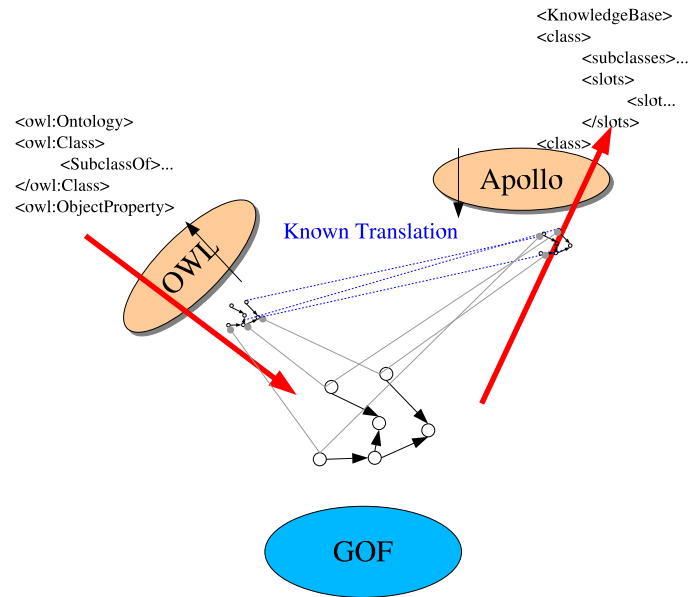


Figure 5.17: Informed transformation

This kind of transformation is called *informed transformation*, because it is *informed* about the source and target formalisms. The name is inspired by the A^* algorithm. Like the A^* algorithm, the *informed transformation* should provide more precise results.³

To distinguish transformation with and without FSOs, the original one (not using FSO) is further called *uninformed transformation* to express, that no information about the original formalism is used.

There are three types of transformation between two ontologies:

Informed Transformation There exists a mapping of the source FSO to the target one. The transformation between formalisms uses FSOs and the mapping between them. Figure 5.17 shows its schema.

Informed Transformation with Mediators There exists (theoretically) a path of formalisms, where for every pair of succeeding formalisms a mapping between their FSOs exists.

Uninformed Transformation If there is no way, how to use FSO, the transformation to the pure GOF is performed. Figure 5.18 shows its schema.

The informed transformation corresponds to the *Mapping approach* mentioned in section 5.3, while the uninformed transformation is generally similar to the *Pivot approach*, but uses simpler language instead of the richest one.

³Providing *informed transformation* with worse results does not make sense.

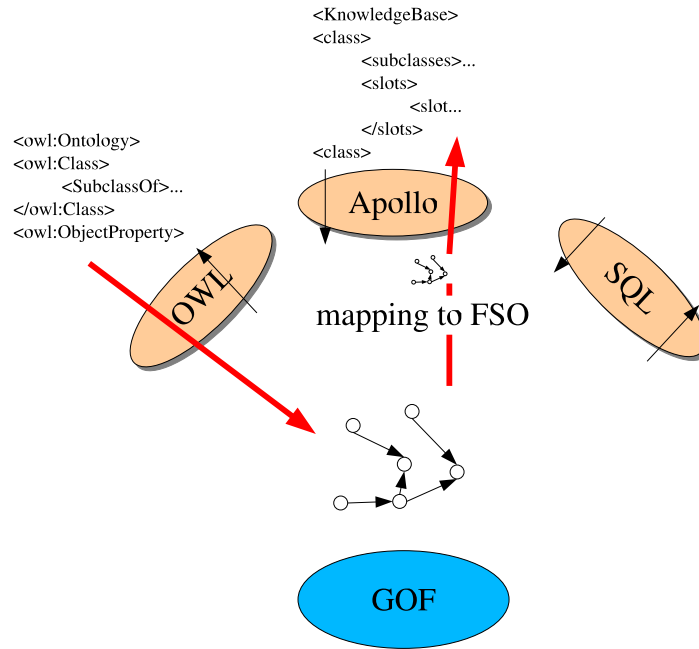


Figure 5.18: Uninformed transformation

5.6.8 Solution for Untransformable Parts of Ontology

As mentioned in the formal definition of ontology, there are two parts of ontology, which cannot be transformed – a set of restrictions \mathcal{S} and a set of actions \mathcal{A} .

The set of restrictions can be stored together with the transformed ontology with connections between rules in \mathcal{S} and corresponding concepts. When transformed back to the original formalism, the restrictions can be reconstructed and verified, whether the result ontology is consistent according to \mathcal{S} .

A processing of the action set is more dependent on the actual form of the actions. Some of them can be processed even in GOF (e.g. actions launched at moment of assigning value to a slot), but for more complex actions (e.g. requiring access to the knowledge base) no solution has been found, yet.

The tested ontologies did not have any action part, which is a rule in majority of practical ontologies. The only available ontology with actions is the time ontology described in [28]. In this case (and probably in most similar situations) the actions do not make sense in a different formalism, as it is tightly tied to the OCML formalism.

5.6.9 Operations

A set of operations can be defined on GOF. The most usable are subset, union, and diff.

The operations are very simple in fact – the design of the formalism allows obvious definition of mathematical operations.

Feature	RDF and RDF Schema	Protégé-2000
Multi-class membership	A resource can be an instance of one or more classes	An instance can have only one direct type
Range constraints	The value of the range property is a single Class which constraints the value of the corresponding property to instances of that class	A value of a slot can be a value of a primitive type or an instance of a class. There can be one or more classes that constrain the value
Containers	There are three types of container objects: bag, sequence, and alternative	Collections have to be encoded, e.g. by ordered lists
Namespaces	Frame names are unique within one schema; for multiple schemas, the XML namespace facility is used to associate each property with the schema	Frame names are unique within one project. Name conflicts are not resolved during project inclusion.
Literal markup	A literal may have content that is XML markup but is not further evaluated by the RDF processor or it can be a primitive datatype defined by XML	Literals can be either plain strings, numbers, symbols, or boolean values

Table 5.4: Summary of differences between the knowledge models of RDF and Protégé-2000

Concepts from two ontologies are considered to be identical if both their namespace and name are equal. Relations are equal if both their arguments are identical and the type of the relation is the same.

Subset of ontology in GOF is simply a subgraph of the corresponding graph. It is frequently used for extraction of a structure of the ontology by selecting only *subclassOf* relation and corresponding concepts.

Union is a combination of two or more ontologies. The operation is equivalent to union of the corresponding graphs.

Diff of ontologies selects the concepts and relations presented in only one of the ontologies. The usage is similar to Unix command `diff`. Mathematically it is equivalent to $(O_1 \setminus O_2) \cup (O_2 \setminus O_1)$.

5.6.10 Common Problems Solved

Similar problems to those met in this work are common problems in using frame-based systems for ontology formalisms based on description logics. An example is the table 5.4 found in article *Using Protégé-2000 to Edit RDF* (<http://protege.stanford.edu/protege-rdf/protege-rdf.html>).

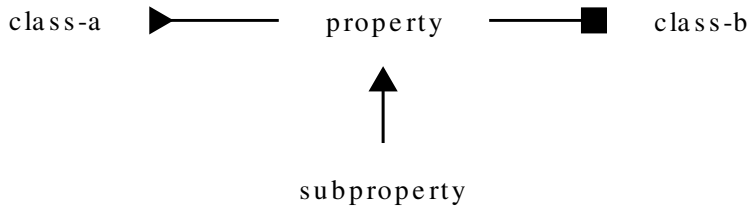


Figure 5.19: Subproperty in GOF

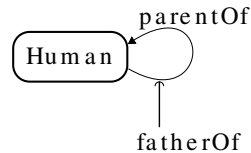


Figure 5.20: Impossibility to draw a graph with subproperty in RDFS

Following sections describe the problems found during translations using GOF formalism. In fact, the problem does not lie in expressing any feature in GOF, but in transforming such a construct into a particular formalism. The structure representing ontology in GOF can change when ontology is being stored into a particular formalism (done by the corresponding gate).

5.6.11 Subproperty Problem

An example of a possible problem during a transformation between formalism based on description logic and frames is the *subproperty* feature (see figure 5.19). Frames do not provide any similar construct – it is not possible to specify a “subslot” as a subclass of a slot. The subproperty makes drawing RDFS knowledge base as a graph impossible. Such attempt is sketched in figure 5.20.

Figure 5.21 shows the running example with a subproperty added. The properties *has-father* and *has-mother* can be generalised and a relation *has-parent* can be their common predecessor. Several kinds of transformation have been investigated.

Although the primarily intended solution has been casted aside, this suggestion used to emerge in discussions, thus it is preferable to explain it here.

This solution assumed structural change – the property splits to a property and a class, the subproperty becomes a class and the property and all instances being targets of such subproperty become instances of the new classes. Figure 5.22 shows resulting schema. Such a solution makes sense – if some concept is a target of a relation *has-father*, it means it is a father. The class *Father* is subclass of both *Man* and *Parent*.

Unfortunately, the solution becomes unusable, when the properties are not valid generally, but express for example personal views. Figure 5.23 shows an ontology with subproperties (*friendOf* and *hates*, subproperties of *knows*), represented in terms of

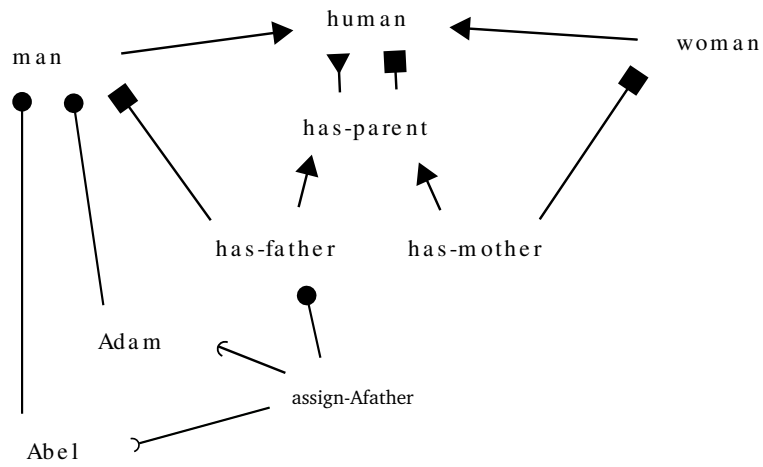


Figure 5.21: Subproperty in the running example

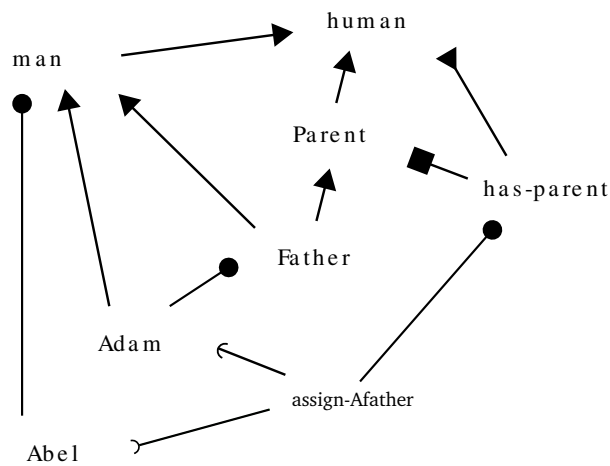


Figure 5.22: First (bad) solution of the subproperty problem

GOF. Applying the prior solution, it leads to classes *Known* and *Hated*. If two opposite views on one concept are present in the same knowledge base, it becomes unclear, who likes the target and who hates it.

The most general solution is in figure 5.25. All properties turn into direct properties of *personS*. Furthermore all instances have to be changed.

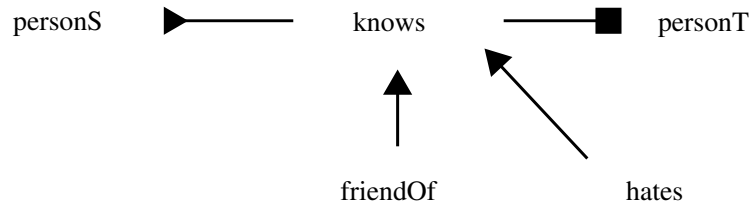


Figure 5.23: Ontology with subproperties

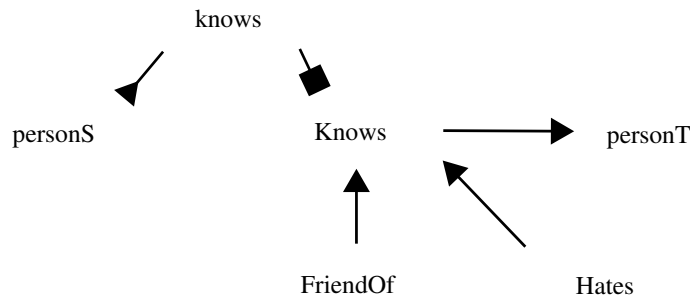


Figure 5.24: Sample Ontology in GOF

Original ontology (from description logic):

$$a \xrightarrow{\text{friendOf}} c, \quad b \xrightarrow{\text{hates}} c \quad (5.5)$$

is changed to

$$a \xrightarrow{\text{friendOf}} c, \quad a \xrightarrow{\text{knows}} c, \quad (5.6)$$

$$b \xrightarrow{\text{hates}} c, \quad b \xrightarrow{\text{knows}} c. \quad (5.7)$$

5.6.12 Instance of Instance

A possibility to have a sequence of *instanceOf* (\rightarrow) relations appears to be a problem for several formalisms. Further problems are connected to models of such ontologies [13].

The approach chosen in GOF is similar to RDF, so the \rightarrow relation is similar to any other relation and can appear relatively freely in a knowledge base.

The GOF formalism have to offer this capability, because it have to cover as wide as possible range of constructs to be able to process formalisms containing them.

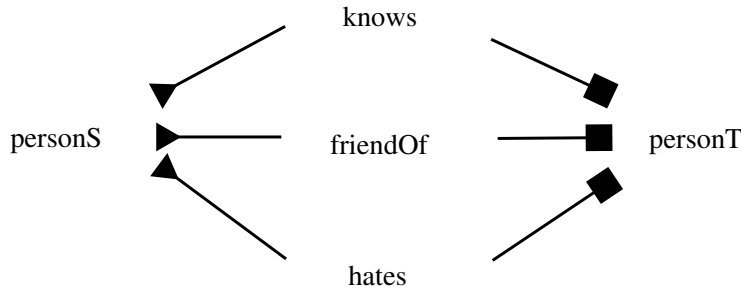


Figure 5.25: Translated ontology

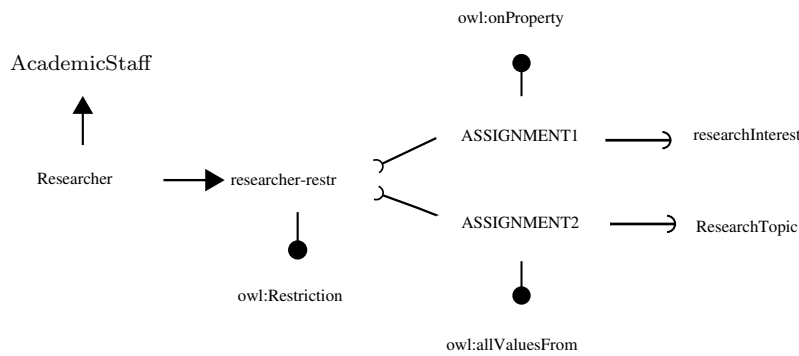


Figure 5.26: OWL restriction model in GOF

Moreover, GOF is working only on syntactic level, it does not define a model.

The \multimap relation have been introduced in order to express decrease in abstractness of a concept. If a formalism does not allow successive instances, it can be replaced by *subclassOf* (\multimap) relation with a similar function (specialisation of a concept). For examples, frames can keep only the last \multimap relation and all preceding change to \multimap .

5.6.13 Restriction Handling

As stated above, the framework focuses on the structural part of the ontology. Therefore GOF was not designed to cover completely the procedural part of the ontology such as axioms, slot facets in OCML and restrictions in OWL. The transformation of an ontology into GOF preserves validity of the axioms and restrictions. When an ontology is transformed into GOF and back to the original formalism, axioms and restrictions can be added to it again. In case the original ontology was consistent, the resulting ontology with added restrictions and axioms is also consistent.

However, in ontology formalisms based on description logics, such as DAML+OIL

and OWL, property restrictions play an important role in definition of classes. Therefore special attention was devoted to them. The example below shows the representation of an OWL property restriction in GOF. The example was taken from the Knowledge Acquisition ontology [19]. The representation in GOF using FSO can be seen in figure 5.26. The closest representation of restriction in frame-based languages is adding a slot with a defined default value to a class. In our example, it means adding a slot `researchInterest` with default value `ResearchTopic` to the class `Researcher`.

```
<owl:Class rdf:about="#Researcher">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#AcademicStaff"/>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#researchInterest"/>
      <owl:allValuesFrom>
        <owl:Class rdf:about="#ResearchTopic"/>
      </owl:allValuesFrom>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

5.6.14 Visualisation

The generalised ontology formalism defined above is capable not only of transforming ontologies between formalisms, but it also allows further processing of ontology in a uniform way. An example of ontology processing is its visualisation.

The visualisation helps human to better understand and manipulate the ontology structure. The author believes, that visualisation in higher dimensions offers more contextual information about the investigated piece of a knowledge base.

George Miller in [32] discovered, that human is able to work with 7 ± 2 concepts at once. This limit have to be taken seriously and only small piece of information should be displayed. Two (or even) dimensions allow to keep connected concepts close together.

Moreover, for humans is spatial orientation natural and could be used for easier navigation in ontology.

A visualisation is used for a survey of unknown ontology, for understanding of its structure and familiarisation with its concepts. The visualisation usually selects only part of whole information (only some relations in our case, the *subclassOf* in most cases). If whole information is displayed, the image ends with too dense graph, messing the concepts rather than clarifying it.

An experience with graphical editors shows, that users tend to create ontology knowledge bases in textual form. A failure of visual editor led to development of Apollo at the Knowledge Media Institute (see section 4.9.1).

Dimensions

The most common way of displaying ontologies in computers is a tree similarly to files are presented in operating systems. In fact, it is a one-dimension (maybe one-and-half) visualisation, as expanded concepts are sorted in one dimension (usually vertically from top to down).

It perfectly serves for ontologies with several concepts. At the moment the ontology grows to several tens concepts, the navigation becomes more and more lengthy and tedious. For hundreds of concepts it is necessary to introduce supporting search – it is impossible to find a concept effectively.

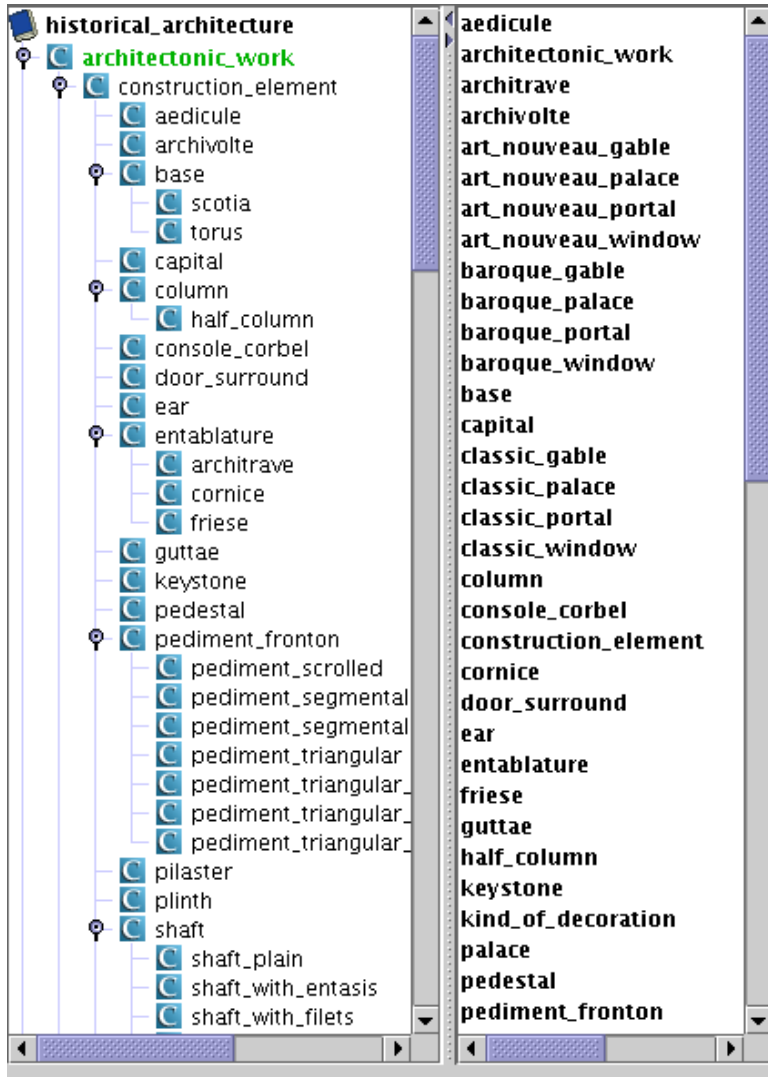


Figure 5.27: Tree (1.5D) visualisation of ontology

Figure 5.27 shows an example from ApolloCH editor. It is clear that for concepts in the middle there is difficult to decide their parents. Even in simple ontologies (under

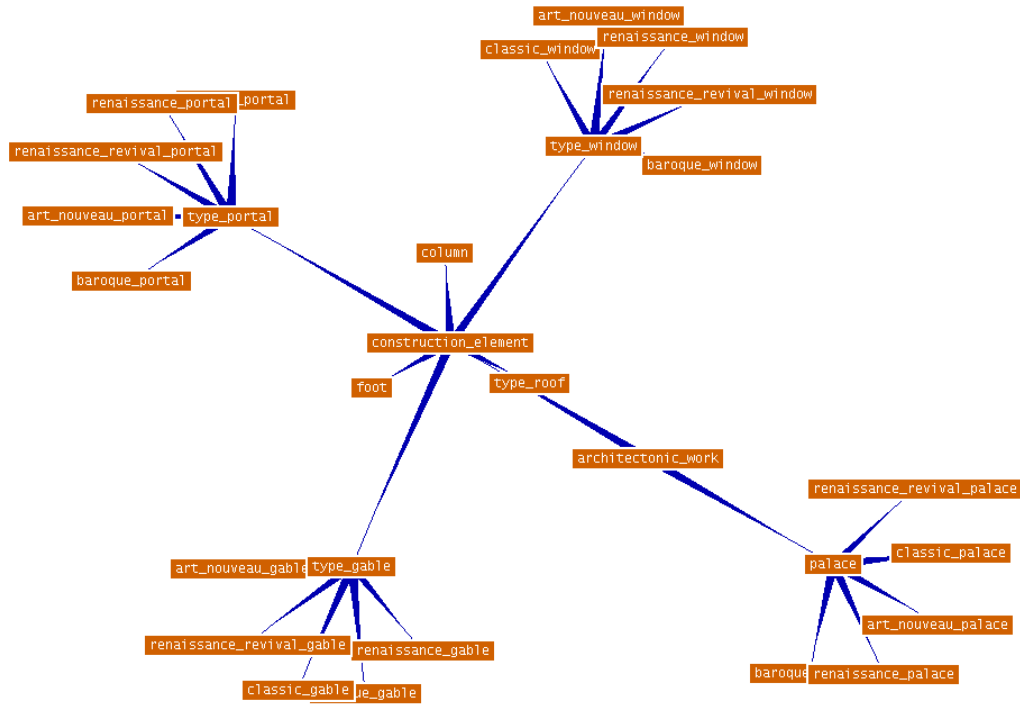


Figure 5.28: 2D visualisation of ontology (TouchGraph)

hundred concepts) is impossible to find a particular concept. This can be tested with the Historical Architecture ontology, mentioned in section 5.8.2.

A better way of presenting an ontology structure is to use more dimensions – two or three. The two-dimensional visualisation provides more natural view of the structure and allows to expand nodes more deeply. The advantage of increased directly visible depth is a better understanding of the expanded node. Figure 5.28 shows an example of a 2D ontology representation.

An orientation in the 3D space is the most natural way for a human. The only drawback is that the current tools for 3D interaction – glasses and gloves – are not enough mature yet. Figure 5.29 shows an example of a piece of ontology.

With the increasing number of dimensions it is crucial how easily the system allows a user to move and to concentrate to the topic of his interest. The system must offer natural physical model of the investigated universe. The way chosen differ from system to system. Some simply zoom into the requested point, while others change the structure to emphasise relations of the point of interest with others concepts. An example of a spatial navigator in a bigger structure is implemented for filesystem (<http://sourceforge.net/projects/xcruiser/>). Further projects come from the sphere of visualisation of biological data.

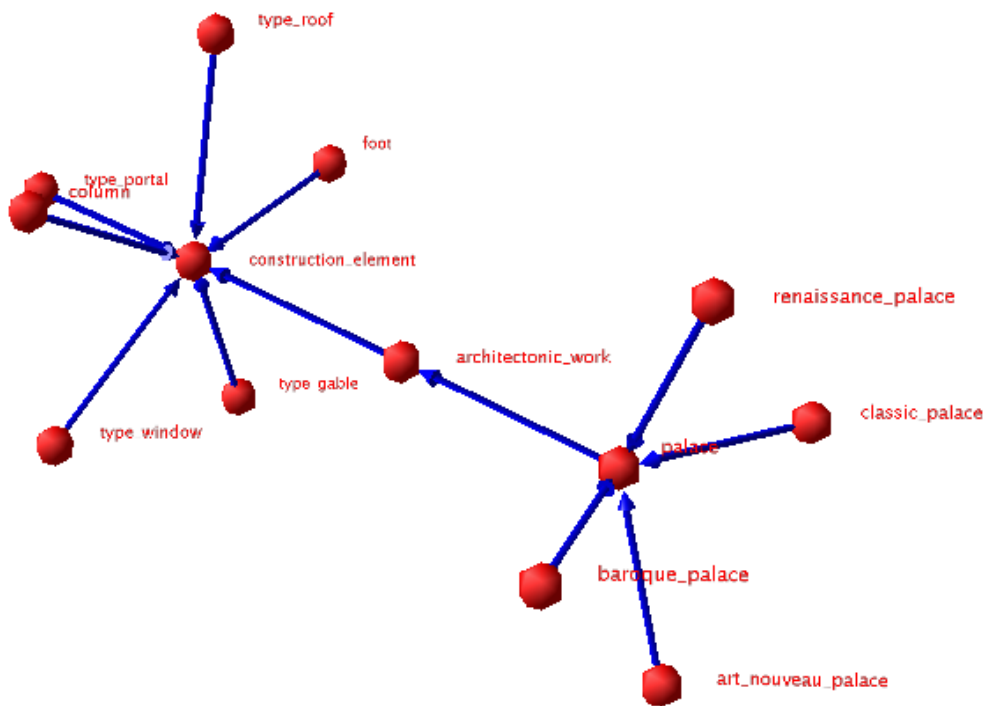


Figure 5.29: 3D visualisation of ontology (Wilmascope)

Visualisation Gates

Within the GOF framework, several gates were implemented to fill models of different visualisation systems and they have been tested on available ontologies.

For displaying the whole graph the main platform is GraphViz suite, especially the dot tool for directed graphs. It offers rich enough set of available arrow types. Unfortunately, the layout algorithm prefers put concepts to one line and dense graphs tend to become a vertical line. For graphs with simpler structure (subclassOf) it gives reasonable outputs. GraphViz generates pictures automatically and is very appropriate for automatic generation of ontology graphs. GraphViz was used for big ontologies presented in the RDFS and OWL frame-specific ontologies (figures 5.14 and 5.15).

The most successful visualisation tried was the two-dimensional TouchGraph visualisation (figure 5.28). It provides living image with the current, investigated concept with its near neighbourhood. The tool allows selection of the deep of the shown information. The best result are achieved with depth two there are displayed concepts connected to the current one by up to two connections. It allows comfortable overview over the context of the concept. A click to some concept moves the context to the new node. The user have the right portion of information all the time. The TG project was used in extended form also for displaying topic maps in Omnigator viewer (www.ontopia.net/omnigator).

A similar approach selects Prefuse. It keeps visible all the nodes, but repositions them as focus moves between nodes. It allows understand the nearest neighbour, but gets messy with a bigger distance.

Hypergraph employes hyperbolic geometry to visualise a (mostly) tree structure. It is an effective way how to survey such structure e.g. *subclassOf* subgraph of an ontology. The nodes too far from the selected one don't show their name, but the structure is still visible. Hypergraph is not capable to handle structures with multiple separate parts and have problems with dense graphs.

There exist also other tools, but it was not possible to try all of them. A set of tools, which will be tested in the future, comprises IsaViz ([\(\)](#)), very nice tool for RDF, Freemind (freemind.sf.net) and others.

5.7 Implementation

The Generalised Ontology Formalism was implemented in order to prove the ideas in the theoretical part of this work. There were written a library with the core of the system and a series of modules prepresenting corresponding gates. Also the mentioned set operations were implemented.

The implementation was done in Java for the intended platform independence. Another strong point in this decision is availability of numerous libraries providing I/O access to the required ontology formalisms and visualisations.

For the OWL formalism two libraries were tested: Jena by Hewlet-Packard and KAON. The Jena library was rather slower and more complex, but it was found to be much more stable and reliable. The KAON library had serious problems with a com-

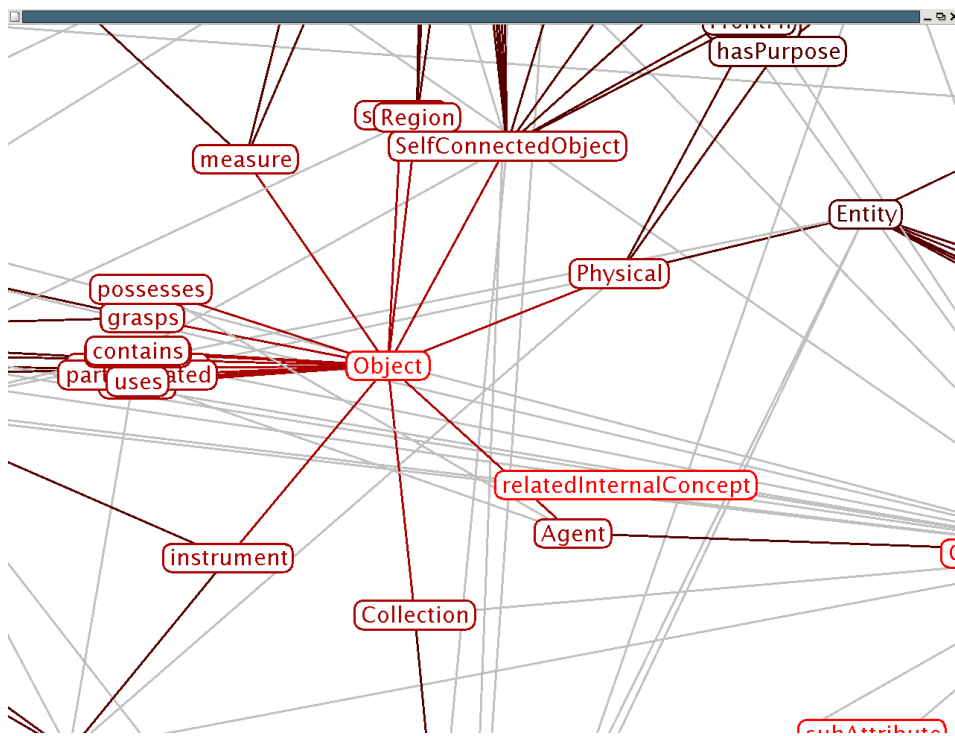


Figure 5.30: Prefuse output in detail



Figure 5.31: Prefuse output – overview

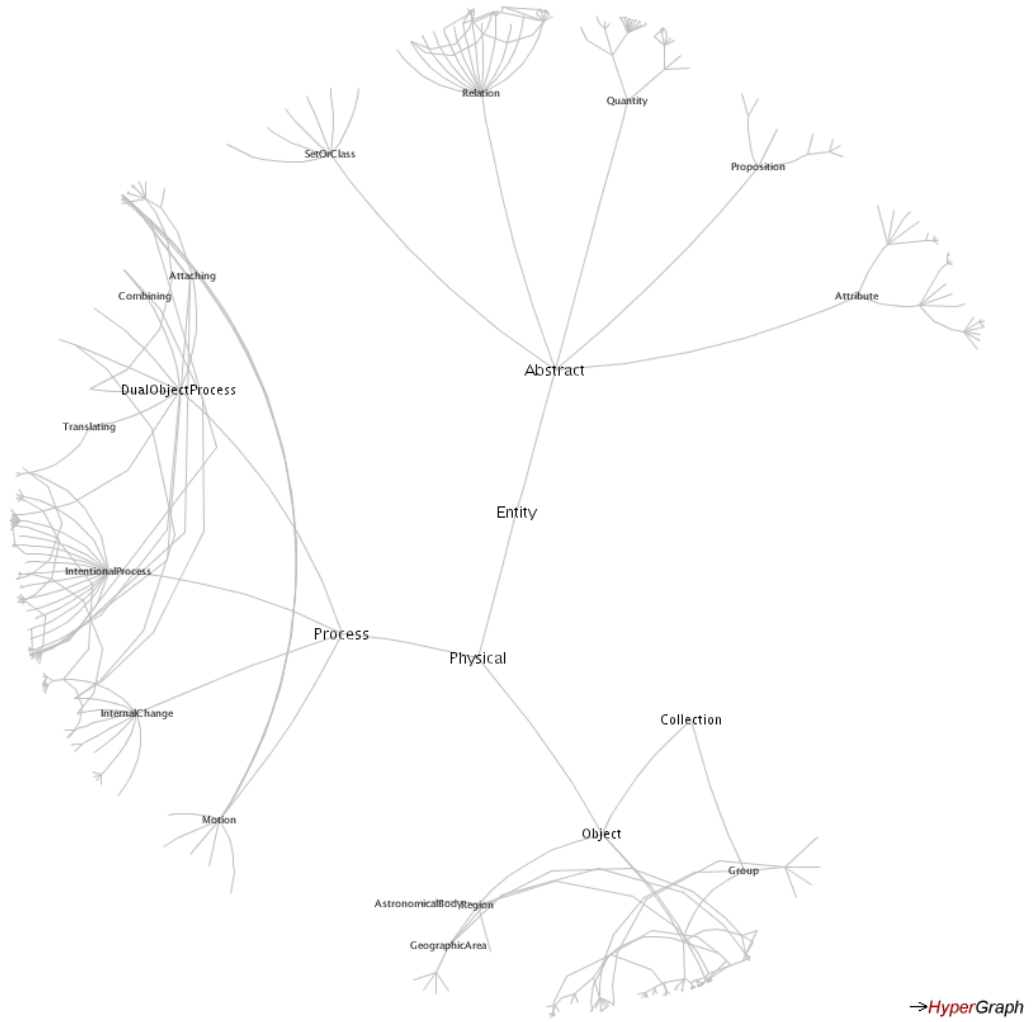


Figure 5.32: Hypergraph output in detail

mon habit in OWL – part of ontologies are written using `OWL:Class` and others with `RDFS:Class`. The `RDFS:Class` were not accessible via KAON and thus are unusable. A similar problem exists with properties.

The OCML formalism was accessed via the Apollo library. Another libraries were used for visualisation as external applications and their names can be found in section 6.3.1.

The GOF implementation can be used from the Java programming language, but is not very comfortable for the end user. The end user tend to use a convenient graphical interface allowing a fast design of the requested actions.

As a such interface was chosen a data processing system SumatraTT designed by the author and developed together with his students and colleagues. More details about SumatraTT can be found in the chapter 6.

5.8 Results

The GOF approach was practically tested. In this section a set of ontologies will be presented together with their transformation and description of the results.

5.8.1 Testing Environment

The time have been obtained as an average value of several measurements. Even though the precise values are not important, only the growth of time with growing number of concepts.

The test run on Pentium IV, 1.7 GHz, 512 MB of RAM. There were active no daemons, only text mode, so whole memory/swap was at disposal to the tests.

During preparation of the test were various external influences eliminated. First, the tests were done several times and the average time has been recorded. Second, as the tests are so called microbenchmarks, their performance depends on an order of test execution. To eliminate the long time of the first execution, the loading tests had “warm-up.” Execution times for both the main ontologies are shown in

The following table shows execution times of loading the trivial ontology in OWL and the historical architecture ontology in Apollo. There is evident exceptionally big value for the first measurement, while the others are reasonably similar. The loading was done repeatedly with and without FSO and the times are show in table 5.5.

5.8.2 Measurements

The set of processed ontologies consists of the following OWL ontologies:

Trivial Ontology is a simple ontology with classes Person, Man, and Woman and a property father.

Wine1 is a subset of the following Wine ontology. It has been prepared for testing purposes (to have a reasonably smaller ontology).

Apollo	OWL
6.288	1.828
1.368	24
915	42
1.256	38
749	22
1.156	17
738	17
1.119	23
740	21
1.137	20

Table 5.5: Warm-up times for Apollo and OWL engines

Wine is a sample ontology accompanying OWL definition.

SUMO (Suggested Upper Merged Ontology) is one of the upper ontologies. A description was given in section 4.10.4.

OpenCyc OpenCyc is another upper ontology (described in section 3.4). Its importance for testing is in its enormous size. This size led into problems in testing (memory exhaust in the Jena or ApolloCH libraries).

Ontologies in the Apollo formalism are only two as this formalism was intended primarily as the target:

Historical Architecture is an ontology developed within the CIPHER project for description of architectural features of building.

CRM is the CRM ontology implemented in Apollo (see section 4.10.5).

From the non-classic formalism was bookmarks, particularly Mozilla bookmarks:

Bookmarks are sample bookmarks in Mozilla format.

Regardless of the intention to get accurate results, the measured times can only be mutually compared. Moreover, only the order of the values should be regarded as the exact times vary in tens of percent.

The transformation of OpenCyc between OWL and Apollo crashed somewhere inside the Apollo library. The problem will be further investigated. At the time of writing this tests, a new version of Apollo library was available, but incompatible with the original way of accessing its XML plugin.

Because the tests with transformation of the OpenCyc ontology were not successful, the whole process was further divided into several tests – loading from source formalism, informed transformation or mapping and the whole process. Measurement of saving was not important, because the times were similar to the loading ones.

Ontology	Formalisms	$\ \mathcal{C}\ /\ \mathcal{R}\ $	Time (ms)
Hist. arch.	Apollo \rightarrow OWL	178/205	1.051
CRM	Apollo \rightarrow OWL	183/281	985
Trivial ont.	OWL \rightarrow Apollo	4/2	47
Wine1	OWL \rightarrow Apollo	72/82	97
Wine	OWL \rightarrow Apollo	713/1.724	4.543
SUMO	OWL \rightarrow Apollo	1.434/1.729	26.458
OpenCyc	OWL \rightarrow Apollo	71.939/85.919	(Apollo crash)

Table 5.6: Whole informed transformation times

Ontology	Formalisms	$\ \mathcal{C}\ /\ \mathcal{R}\ $	Time (ms)
Hist. arch.	Apollo \rightarrow OWL	178/205	976
CRM	Apollo \rightarrow OWL	183/281	903
Trivial ont.	OWL \rightarrow Apollo	4/2	37
Wine1	OWL \rightarrow Apollo	72/82	99
Wine	OWL \rightarrow Apollo	713/1.724	2.531
SUMO	OWL \rightarrow Apollo	1.434/1.729	35.560
OpenCyc	OWL \rightarrow Apollo	71.939/85.919	(Apollo crash)

Table 5.7: Uninformed transformation times

The first table, 5.6 shows times of the whole informed transformation and table 5.7 the same for uninformed transformation.

The other two tables 5.8 and 5.9 show only the loading times. Although the time of loading of OpenCyc is very long (circa one and half hour), parsing of the input file by the Jena library was quite fast – about tens of seconds. The rest is spent in processing of the tree and in not optimised internal structures of GOF.

The long times, especially in processing of OpenCyc are due to non-optimal loading of the OWL formalisms, which connects various concepts and thus intensively searches in the knowledge base. This part will be optimised in the future..

As an independent part was tested an informed transformation. A knowledge base was pre-loaded and not saved, so only the transformation was measured to be able to compare with to the corresponding mapping in uninformed transformation. The table 5.10 shows results.

For uninformed transformation, the mapping into the target FSO is important. In the current implementation, the mapping engine is used. Although it has been shown it is an NP problem, the achieved results are satisfactory. The table 5.11 shows timing of the mapping engine.

In both informed and uninformed transformation can be seen, that the *Wine* ontology is more demanding than the *OpenCyc*, although the *OpenCyc* is 50 times bigger. The reason is a complexity of the *Wine* ontology. It has been developed as an example

Ontology	Formalism	Concepts $\ \mathcal{C}\ $	Relations $\ \mathcal{R}\ $	Time (ms)
Hist. arch.	Apollo	178	205	999
CRM	Apollo	183	281	742
Bookmarks	Mozilla	148	147	1.832
Trivial ont.	OWL	4	2	160
Wine1	OWL	72	86	154
Wine	OWL	713	1.728	771
SUMO	OWL	1.434	1.779	1.946
OpenCyc	OWL	71.939	85.919	6.755.855

Table 5.8: Time of ontology loading without FSO ($\phi_e = \emptyset$)

Ontology	Formalism	Concepts $\ \mathcal{C}\ $	Relations $\ \mathcal{R}\ $	Time (ms)
Hist. arch.	Apollo	182	363	1004
CRM	Apollo	187	464	804
Trivial ont.	OWL	24	8	21
Wine1	OWL	107	188	357
Wine	OWL	1.022	2.637	906
SUMO	OWL	1.454	2.647	2.959
OpenCyc	OWL	71.959	116.339	9.227.209

Table 5.9: Time of ontology loading with FSO ($\phi_e \neq \emptyset$)

Ontology	Formalisms	$\ \mathcal{C}\ /\ \mathcal{R}\ $	Time (ms)
Hist. arch.	Apollo \rightarrow OWL	178/205	1
CRM	Apollo \rightarrow OWL	183/281	1
Trivial ont.	OWL \rightarrow Apollo	4/2	3
Wine1	OWL \rightarrow Apollo	72/82	14
Wine	OWL \rightarrow Apollo	713/1.724	2.010
SUMO	OWL \rightarrow Apollo	1.434/1.729	14
OpenCyc	OWL \rightarrow Apollo	71.939/85.919	476

Table 5.10: Informed transformation times (without loading and saving)

Ontology	Target Formalism	Concepts $\ \mathcal{C}\ $	Relations $\ \mathcal{R}\ $	Time (ms)
Hist. arch.	OWL	178	205	1
CRM	OWL	183	281	1
Trivial ont.	Apollo	4	2	0
Wine1	Apollo	72	82	8
Wine	Apollo	713	1.724	121
SUMO	Apollo	1.434	1.729	31.976
OpenCyc	Apollo	71.939	85.919	1.253

Table 5.11: Mapping times

ontology and contains many advanced features to emphasise various features of OWL. On the contrary, OpenCyc is a production system and contains only few such features.

5.9 Conclusion

The Generalised Ontology Formalism provides both theoretical and implementation framework for ontology transformation between formalisms. This formalism is used as a dictionary in the process of information migration between different systems for both data conversion and information interchange between the systems.

The relative simplicity of GOF makes it possible to develop transformations to/from many formalisms and share ontologies from multiple sources in a uniform way.

GOF also makes it easier to perform operations on ontologies based on either equal or different formalisms such as determining differences between ontologies, merging ontologies etc.

6 SumatraTT

The author of this thesis concentrated his efforts on data preprocessing during his PhD study and mainly on a universal system for data transformations. Later he concentrated on usage of ontologies and translations of ontologies. The research lead to development of the system SumatraTT 2.0.

SumatraTT [2, 11] (<http://krizik.felk.cvut.cz/sumatra>) is a universal data processing system, developed at Department of Cybernetics, CTU in Prague. Its primary purpose was to provide a modular platform allowing module developers to concentrate on their specific task while generic activities such as data loading, saving, various transformations, and graphical representation are supported by a standard set of modules.

The system was designed as general data processing system originally aimed at data preprocessing for data mining and data warehousing.

A screenshot can be seen in figure 6.1.

6.1 History of SumatraTT

During work on GOAL project (Geographical Information Online AnaLysis, [30]) there was necessary to process raw data obtained from sensors measuring level of water in water tanks and several other sources with temperatures, weather type etc. Data had a format unreadable by available tools (MS Data Services for SQL Server, text drivers for ODBC).

A small application in C++ was prepared for this simple task. Later, as required operations become more complex, the engine was made general with an execution core and a set of drivers for various data sources.

The data sources were defined by metadata describing type and connection details (filename, connection to database, passwords etc.).

6.1.1 SumatraTT 1.0

Approximately at this time project Sol-Eu-Net started. A package for data preprocessing have been implemented at the Czech Technical University within its frame.

The most important feature added was internal scripting language with a syntax very similar to Java. Because of the similarity, both language and the system have got the name Sumatra. Later, in order to distinguish the language and system, the language was renamed to SumatraScript and the system to SumatraTT.

Writing a special programs in SumatraScript didn't bring any advantage. The power of embedded scripting language showed, when a library of templates were introduced.

The final program was automatically generated from a template and a description of processed data sources.

The description of required transformation (used template, details of the transformation) become a part of the metadata.

The system started to be used by external users and the metadata coding become a problem. For such users has been developed a graphical user interface. It showed a list of available templates, a list of data sources, and a pane for design of the transformation. This interface allowed using of SumatraTT by non-expert users. As an implementation language was chosen Java partially for platform independence, because the core was going to be ported to Linux, and partially for its good design of user interface controls.

6.1.2 SumatraTT 2.0

At the end of the Sol-Eu-Net project, it was clear that for common users the idea of separated data sources, templates, and metadata is too complicated and they are not able to fill correctly transformation parameters as it required reading manual.

Also the scripting language become a problem. Only one developer¹ learned SumatraScript and was able to write a template.

The author of this work decided to change the idea of metadata driver transformation and designed module-based framework. Later, he implemented the core and wrote several basic modules. The whole system has got name SumatraTT 2.0.

Again, for platform independence, the Java programming language was chosen. For this decision were more reasons – clear advanced access to databases (JDBC compared to ODBC), multithreading as an integral part of the language, easy application interoperability, many available and free libraries, etc.

The choose of the language together with the modular design showed to be the best. Today, in SumatraTT 2.0 there are modules implemented by different people.² Especially for diploma thesis is SumatraTT ideal framework were students can show their theories working and their modules can enrich the capabilities of the whole system. Also malfunction modules can be easily avoided.

SumatraTT 2.0 is nowadays a user-friendly modular system for general data processing, aimed mainly at data preprocessing for data mining and data warehousing (Extraction, Transformation and Load process, ETL). It consists of an execution core, a graphical interface providing a platform for defining the transformation, and a set of modules solving various tasks of data processing. Some modules cover generic tasks and there are also groups of very specialised modules targeted to specific problems.

The graphical interface helps the user to quickly build the intended transformation without any programming, as the set of available modules is sufficient for most applications. In rare cases when the task goes beyond the scope of prepared subtasks/modules, it can require simple scripting.

¹Steve Moyle

²Besides Petr Aubrecht, the most important modules were written by Martin Glogar and Lenka Nováková. Few modules were done by various students.

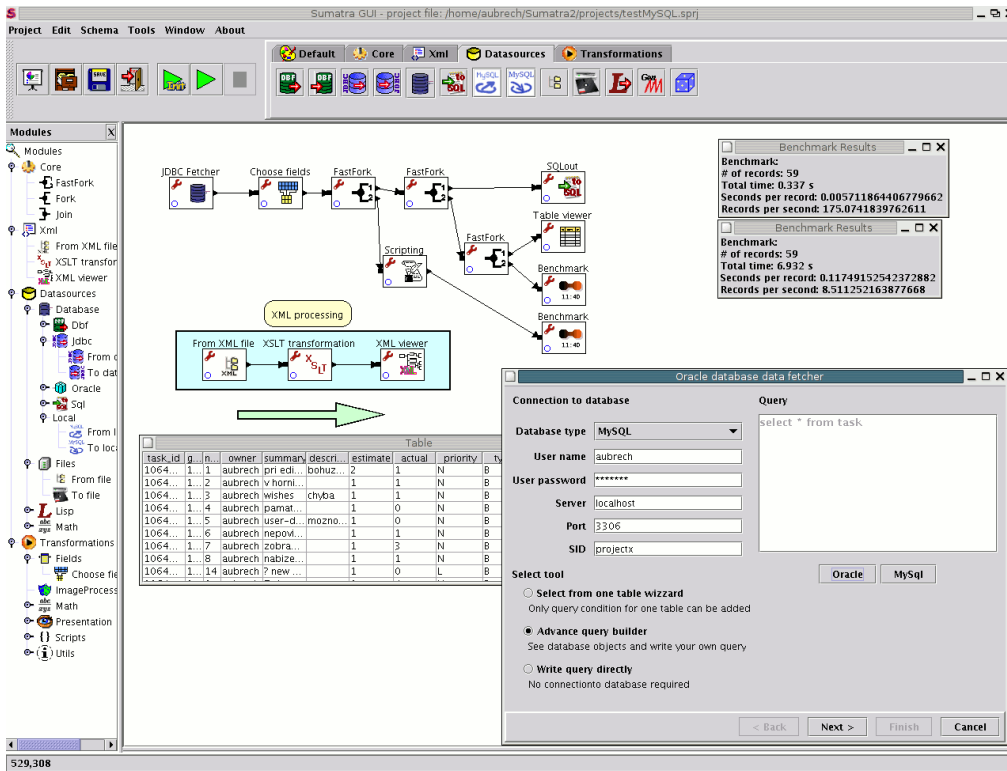


Figure 6.1: SumatraTT 2.0 screenshot

The idea of interconnected modules is similar to that used in the original SGI Explorer (with about 150 modules, http://www.nag.co.uk/Welcome_IEC.html) and its solution is very popular in many modern systems.

In SumatraTT, the most preferred approach to data processing is streaming – one record is loaded, fully processed and saved. In this way huge data can be processed. An exception to this rule are modules requiring all data available and thus appropriate for smaller datasets (for example dynamic graphs). As a compromise solution, the concept of an internal database has been introduced. It means that data is first stored temporarily to the internal database, which is further processed by means of SQL.

SumatraTT is an open system with GNU Public License built on open technologies – including popular GPL projects JFreeChart, BeanShell scripting, or McCoI database. Using such libraries brings fast development. Moreover, the libraries are growing independently on development of SumatraTT. This way, the whole system is being actually developed by a number of developers.

The openness together with modularity (especially independence between modules) enables creating a wide scope of modules, from which it is possible to choose appropriate ones best fitting particular needs. Thus, modules of insufficient quality can be easily omitted, while the best ones become a part of the official release.

The whole system offers a base of Pyle's idea of Prepared Information Environment introduced in [42].

6.1.3 Graphical Interface for Data Preprocessing

SumatraTT 2 development is highly concerned with user's convenience whenever possible – it offers e.g. automatic creation of project documentation or opening the most recently edited project by default, see section 6.2.1. Why there is so strong emphasis on convenience?

There exist several ways how to do data preprocessing. Experienced users tend to use their rather low-level tools like SQL scripts, Perl, etc. Such a low-level processing offers high performance, but it has serious drawbacks, namely rather high efforts spent on script development and a rather difficult maintainability of such scripts by other people.

On the other hand, graphical user interface (GUI) makes the process of preparing a transformation very fast, easy-to-debug, and easy-to-maintain. It also simplifies supplemental activities like documenting the transformation or deployment of the process to multiple computers carrying out the transformation in parallel.

SumatraTT has been developed with an emphasis given to both the easy-to-use graphical interface and powerful features provided by a wide scope of modules ranging from simple to very specialised ones. Each user can choose modules appropriate to his/her level of experience. In this way, it is possible to offer both the flexibility of graphical systems and the power of scripting languages with only small limitations.

One of the aims of SumatraTT is to bring data preprocessing tools closer to non-expert users who tend doing data preprocessing in restricted tools like spreadsheets as well as to the users repeatedly writing single-purpose programs.

SumatraTT can be also used by experts as an experimental environment, and as a test and verification platform allowing to design appropriate transformations, which can eventually be re-implemented in a generic programming language.

An advantage of using SumatraTT is its uniform way of data representation, which allows integrating it with a number of related tools providing e.g. data visualisation and understanding. Future development plans include developing of inter-operability with external application via web services, CORBA, etc.

Selected techniques making user's work easy and fast:

- modules providing similar functionality are represented by similar icons,
- re-opening the most recently opened project on start by default,
- indicating progress of the transformation process on processing each record,
- several ways how to a module can be inserted,
- the project workspace can be enriched with user comments and graphics to document the technique used (these components will also appear in the generated documentation),
- automatic documentation function generates a complete documentation of the project,
- use of drag&drop.

6.2 Features

6.2.1 Basic Concepts

SumatraTT consists of a core and a library of modules.

The core is a shell, which allows designing a transformation by putting icons representing individual modules onto a canvas and defining data flows between modules simply by drawing lines between respective icons, setting up modules' properties and managing the execution of the whole data transformation process. Moreover, there are wizards at disposal, which are targeted at specialised tasks, like adding many connections to database tables in a single batch, documentation, definition of groups, etc.

Modules represent elementary tasks and are grouped by their functionality in a tree-like structure. Modules cover all important areas of data processing. For the complete list of available capabilities see section [6.2.5](#).

Modules implement rather simple actions, which the user can combine in order to achieve the desired transformation. Such a split of complex functionality speeds up the development, simplifies debugging and allows a number of ways how to combine modules in order to get the required functionality.

The design of SumatraTT strictly separates two phases of a data transformation : transformation design and execution. Some other data preprocessing tools for data mining join both phases, assuming the amount of processed data is small enough and the processing is fast. SumatraTT is expected to process large amounts of data (it has been used to transform hundreds of megabytes of CSV files) and the transformation can take for hours.

In the design mode, the modules are placed on the working area (canvas), interconnected, their parameters are set, and optionally documentation pieces are written. Modules can be grouped into components in order to divide the whole project into logical parts. Every component can be run separately.

In the processing mode, the prepared schema is executed. SumatraTT offers two ways, how to run the project:

- from GUI, with convenient process supervision, or
- from the command line. This mode allows a delayed run (e.g. over night) and/or a client/server configuration with clients sending project definitions to the server to be executed remotely there.

Usually, the project is executed in the GUI environment on the same workstation, on which it was designed. This approach is appropriate for the majority of application with exception of processing a large amount of data.

An important design feature is the negotiation of a data structure done at the beginning of the transformation process, not necessarily during creation of the transformation schema. In such a case, the data structure in use is not restricted to that one known at

the design time, but modules negotiate the most appropriate one and adapt themselves to it. The negotiation is described in section 6.2.3.

SumatraTT can also cooperate with other applications. It can either launch another application, provide it with data and collect results, or it can be itself launched by another application in a silent mode without the GUI, only processing data.

According to the CRISP methodology, the data preprocessing phase takes a significant part of a DM process. It itself consists of two steps: data understanding and data preparation. In fact, both steps form a cycle, when data is iteratively preprocessed and visualised to reach better understanding, which leads to another (modified) preprocessing and visual output and so on until some condition has been reached.

This cycle can be of two kinds with regard to the condition stopping the iteration. Either it is possible to calculate the condition – then it is easy to incorporate the cycle directly into the transformation schema which can cycle without user’s interaction, or it is not.

If we are not able to specify the appropriate stopping condition (say, “until the data show some error values”) it is possible to divide the schema into parts and create groups of modules. Each part can be executed independently and the iteration can be controlled by user.

6.2.2 Advanced Features

SumatraTT system includes several advanced techniques used to extend its functionality.

One of the most useful features is SumatraTT’s autodocumentation capability. The system automatically creates a complete documentation of the project schema, including the overall schema and detailed settings of each module. User can optionally describe textually selected parts to improve the documentation and make it more descriptive. For further information see section 6.2.4.

SumatraTT makes it possible to wrap up known good solutions into so called Best Patterns.³ A part of the system is a library of ready-to-use solutions of particular situations called patterns. Patterns are parts of transformation schema consisting of pre-setup modules prepared by experts to solve particular tasks. The user can select a pattern according to his problem and the corresponding solution is then automatically inserted into the current transformation schema.

A common approach has been chosen to fill potential gaps in functionality by introducing a scripting capability. Specific extensions of this usual technique is described further in section 6.2.5.

SumatraTT has a special support for SQL databases, which can be accessed directly by some modules, taking advantage of the SQL query language, many data miners are familiar with. Data can be sorted, grouped, etc. Moreover, SQL enabled modules may provide various statistics or algorithms, which need random access to data.

For projects requiring work with statistics and other operations which would be easily done using SQL, but having data in plain text files, SumatraTT offers its internal

³Best Patterns is a similar to Best Practices used in data mining.

database. There is a way how to store data in its private data storage and work with it without the necessity to install and set-up a connection to any external database. The database engine is powered by the open source Mckoi database.

Besides generic functionality, in SumatraTT there are areas dedicated to specialised topics. An example of such an important topic is the support of data mining process and its typical tasks. One of them is making various types of subsets – fair subset, vario subset, etc.

Neither structured documents can be avoided and SumatraTT has a part dedicated to processing XML documents. The capabilities are described in section 6.2.5.

6.2.3 Stream design

The processing schema represents the data flow of the transformation. The schema consists of modules and connections between them. Both parts are described in the following sections.

The most important design decision in SumatraTT internal architecture concerns with the architecture of connections. However, it is necessary to start with explaining the architecture of a module.

Module Architecture

The hierarchy of modules shown in the user interface exactly corresponds to hierarchy of directories within the `modules` directory.

A module consists of one jar file (Java ARchive) in directory `modules`. The file contains a configuration file with complete parameters of the module, its Java classes, and accompanying files (icons, etc.).

The parameters in the module configuration file include the name of the module, its documentation, number of inputs/outputs, names of icons, and names of Java classes the module is composed of.

There are up to three classes for one module: Processing, GUI, and Presentation.

The Processing class must implement the `ModuleInterface` system interface. It is the only required class per module and represents the execution side of the module.

Most modules need to be set-up by the user. This is task for the GUI class. It is simply a `JFrame`, which is shown, when the user wants to set-up module parameters by means of the GUI. The GUI class is optional – if it is not present, SumatraTT will ignore user's actions aimed at module set-up.

The last class is intended to represent the module directly in the design area. It can show process of the transformation within the schema. If not present, the module is displayed as a box with the corresponding icon.

Connection Architecture

SumatraTT is designed to process huge amount of data. To limit the consumption of operational memory, only a small data element is kept in memory. Such a data element

is a vector of attributes similar to an SQL record. The processing is carried out record by record – from fetching the record from a dataset, through transforming the record, to storing the result a dataset. This approach allows unlimited size of data being processed and it is appropriate for most of transformations. Special modules or group of modules are at disposal for specific cases when this approach is not appropriate.

The record is stored into the connection between two modules. The source module sets the content of the record and passes the control to the target module, which reads the data from the connection.

The structure of the record is done within intermodule negotiation. The source module sends message with the proposed structure to the target module, which can reject it. A modified structure can be send, until both modules accept the final structure.

A special technique is used in order to decrease of necessity to copy data between modules. The record attribute belonging to a connection can point to another attribute from different connection. This shares the data between connections, usually between two connections going to and from a simple module, working with only one attribute. The others are shared and thus do not need to be copied. Therefore modularity does not require massive copying and does not impact effectivity of the transformation.

SumatraTT supports various attribute formats from the usual ones (Integer, Double – real number, String) to more special (Xml, Image, binary objects). The binary objects use a table architecture to share Java objects between modules for special purposes. The field handling include missing values like NULL in SQL databases.

Metadata

The connection description so far concentrated on the data side of transformation. In processing in SumatraTT there is used also metadata. It has always a form of a small XML document. Metadata is used for three purposes:

description of the whole dataset – record structure, statistics, etc.

record description – description of one particular record

commands – start/stop, eof, restart, user-defined commands

To serve all kinds of metadata, the connection in fact consists of two channels – data and metadata. It is possible to use only one channel or both together. Descriptions and commands are sent using only the metadata channel. When data is processed, the metadata can be used for description of the particular record. For a detail of the schema see figure 6.2.

An important feature helps to create new commands – messages unknown to the module are sent further. This way the message can pass modules between the source and the intended target without change.

The graphical interface also uses the metadata commands for communication with modules. For example in the beginning of transformation the following sequence of messages is sent:

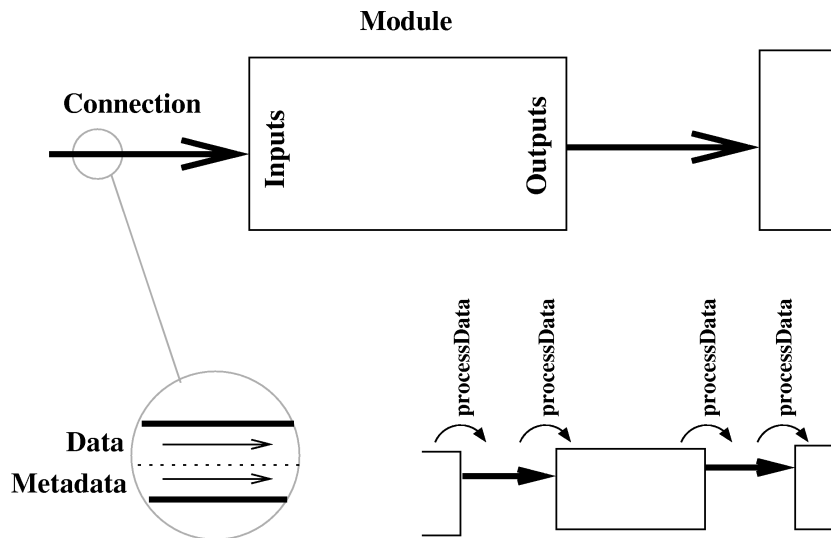


Figure 6.2: SumatraTT 2.0 Structure

<**initialize**> – initialise the module,

<**prepareformat**> – negotiate the record format between modules,

<**start**> – start processing.

The sequence can also run without the <start> command. It is used in so called “Init Run” – some modules need to know the record formats in order to provide right user interface (for example field selection module).

When a module finishes, it sends <eof> message. If the user decides to interrupt the transformation by pressing the Stop button, system sends <stop> message.

A connection is tied also to the thread model of SumatraTT. The system uses threads to parallelise processing of parts of the schema. Especially for transformations working with multiple database connections is parallel processing an advantage.

Instead of launching a new thread for each module with an expensive synchronisation, SumatraTT splits the schema to parts and sequential parts run in one thread. It removes the synchronisation and makes the transformation almost as effective as writing the code in pure Java.

6.2.4 AutoDocumentation

A documentation is an indispensable part of each project and often takes a lot of time. Data preprocessing has to be documented for future reusing and data understanding. When data quality becomes uncertain, it is necessary to check how the data was transformed and which methods of preprocessing were used.

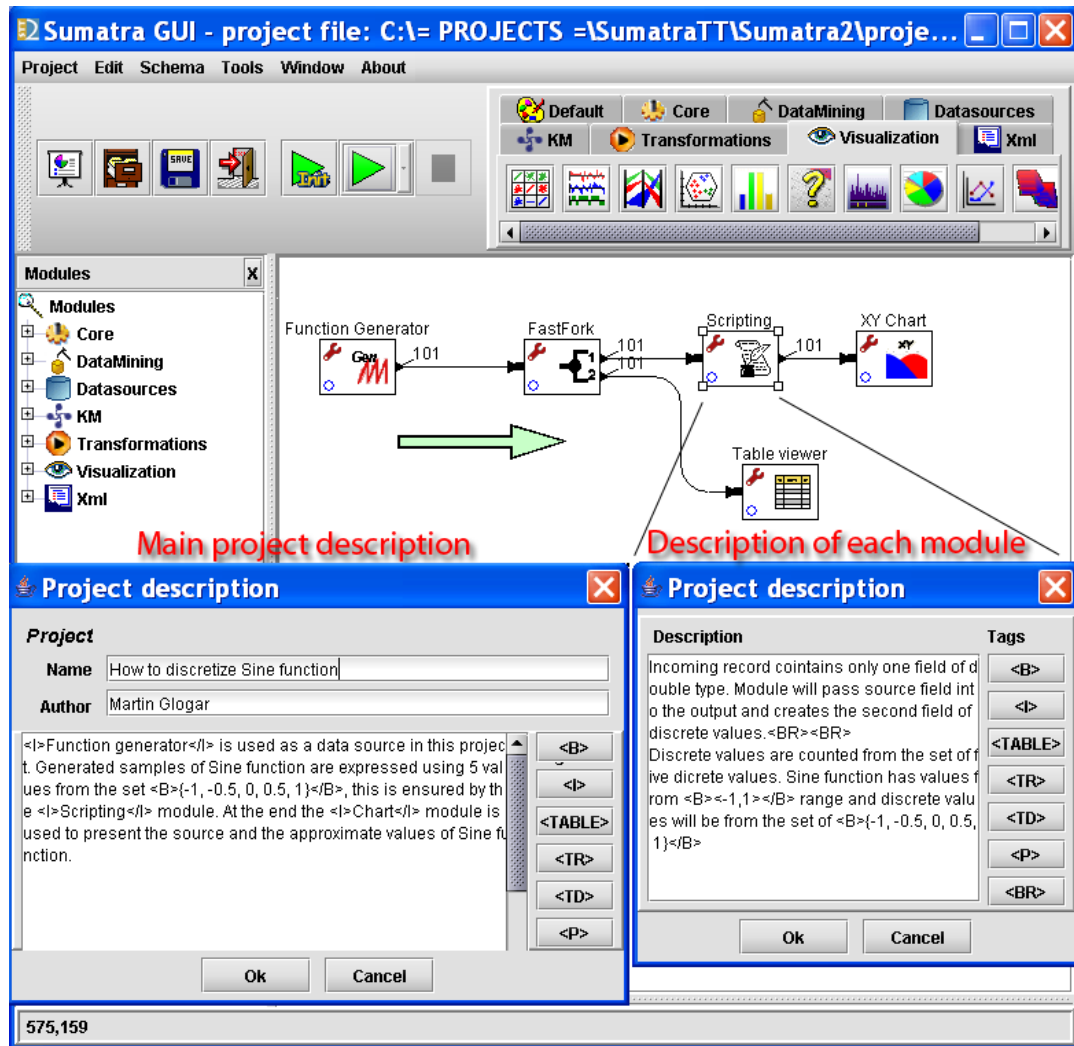


Figure 6.3: Support for documentation in transformation schema

A HTML documentation generator was implemented to help with documentation both globally on transaction diagram (project description) and individually for each module to describe function of used module and its contribution for whole transformation process.

The documentation contains by default all module settings. User can improve the documentation by including comments to both schema and module description. This piece of information describes the function in the transformation schema (see figure 6.3).

The information generated automatically about the module settings is given by the module author (it can for example contain a description of the module behaviour).

Documentation is generated in the HTML format in multiple files for simple navigation between documentation details and easy distribution and presentation (see figure 6.4). The left side contains an index with links to all used modules. The main part displays the description of the whole schema (again with links to the modules details) or the details.

Project

[Description](#)

Modules

[FastFork](#)

[Scripting](#)

[Function Generator](#)

[XY Chart](#)

[Table viewer](#)

Sumatra 2 project documentation *Sumatra 2*

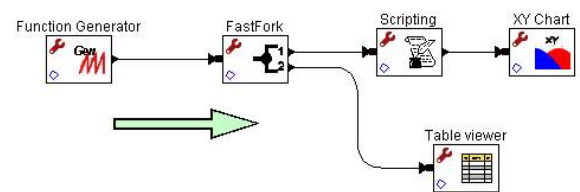
[Czech Technical University](#) - [Faculty of Electrical Engineering](#) - [Department of Cybernetic](#)

Project description

Project name: How to discretize Sine function
Author: Martin Glogar

Function generator is used as a data source in this project. Generated samples of Sine function are expressed using 5 values from the set $\{-1, -0.5, 0, 0.5, 1\}$, this is ensured by the *Scripting* module. At the end the *Chart* module is used to present the source and the approximate values of Sine function.

Project schema



```

graph LR
    FG[Function Generator] --> FF[FastFork]
    FF --> S[Scripting]
    S --> XC[XY Chart]
    S --> TV[Table viewer]
  
```

Project modules

Figure 6.4: Project documentation in HTML format

6.2.5 Available Modules

The present set of SumatraTT modules is rich. The following section describes rather groups of modules; description of all modules is a task for a documentation with many pages.

The main focus of SumatraTT is targeted to the data processing and input/output modules play the most important role. SumatraTT offers a wide range of supported formats to read from or write to. The most important is access to relational (SQL) databases and a plain-text files. Both have a comfort setup to tune the connection to the least detail. Other formats include the DBF standard, Excel spreadsheet, WEKA format for data mining tool or SQL INSERT commands. There are modules for artificial intelligence application generating Prolog source or reading a Lisp source as structured data. An important part of the library are XML modules. And finally there are very specialised approaches like the group of 22 modules processing different ontology formalisms.

After the data is loaded, the generic transformations come into use. This group of modules includes making subsets of both records and fields, joining them, mathematical operations, and support modules for project debugging and benchmarking.

For data mining, there are modules dedicated to cluster analysis and time series and others are being developed.

Some groups are described in their own sections: scripting support in the section 6.2.5, XML in section 6.2.5, visualisation in 6.3.1.

Besides the large set of general modules there are sets of special modules targeted to

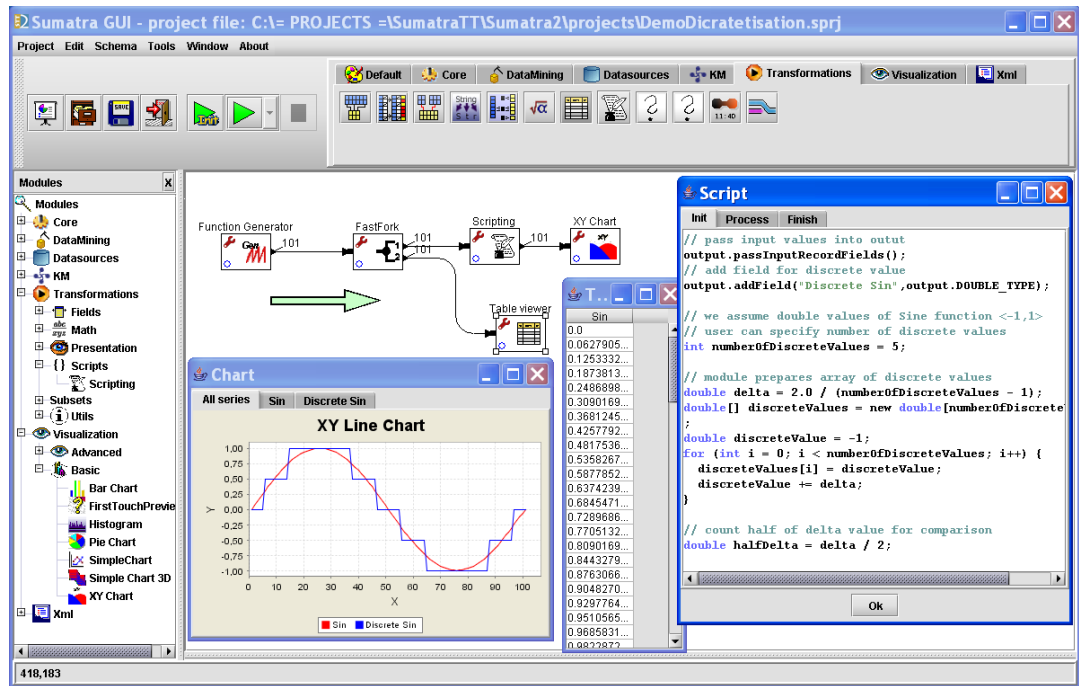


Figure 6.5: Example of scripting in SumatraTT

a narrow domain. An example is knowledge management domain, which contains 22 modules for transformation between different formalisms and supporting operations.

Scripting Support

Although the number of modules is big enough for most application and still grows, it cannot cover every small detail requested by the task. These gaps solves a scripting support. The scripting module allows to implement the functionality demanded in Java language. The most standard approach has been chosen – the scripting language is BeanShell with syntax of Java 5. The BeanShell mediates access to all internal structures and libraries, allowing in this way arbitrary action to be performed like native modules written in Java.

In the future there will be possible to automatically generate a Java source of a module using the script. A developer will implement the task in script module in a short development cycle, quickly test its behaviour. After verification the content of a new module will be generated, compiled and included in SumatraTT library.

There is also planned a support for other languages (Python, Perl, etc.) with libraries available.

For a small example see figure 6.5. Function Generator represents data source and generates samples of Sine function of range $(-1, 1)$. Scripting module is used to discrete samples into 5 values from set $\{-1, -0.5, 0, 0.5, 1\}$.

Module contains three parts of script:

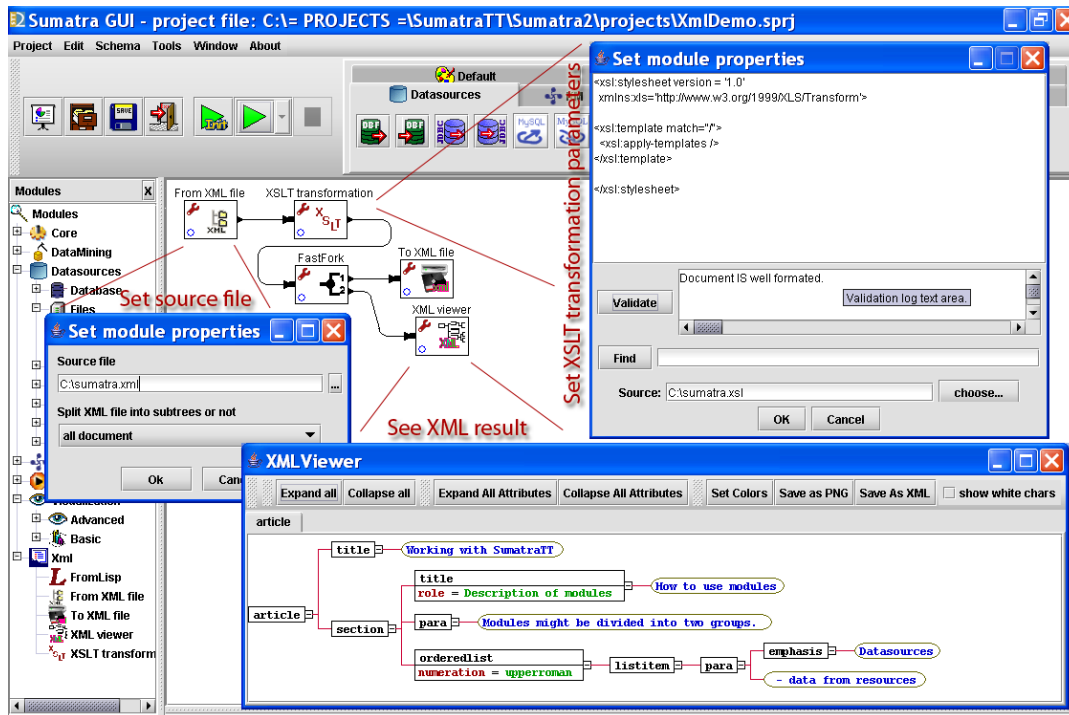


Figure 6.6: SumatraTT 2.0 XML processing

Init – initialisation part specifies module output format (called one time – used in negotiation between modules when transformation process is initialised) and all next necessary initialisation of variables used in transformation process.

Process – processing part is responsible for transformation process. Script is called each time when data record arrives into module. Incoming fields from input gate are used as data source and values of output fields are set at the end of script. When processing script is done, output record is sent to the next module as data flows in SumatraTT transformation schema.

Finish – this part is called one time at the end of SumatraTT transformation process. Sometimes user might demand to do anything when SumatraTT transaction finishes.

For the result of a transformation see figure 6.5. Chart presents behaviour of a Sine function (source from generator) and its discrete behaviour.

XML

Processing of XML documents cannot miss in any preprocessing system as the popularity of this format increases rapidly. In SumatraTT the XML format is supported internally as a standard type of field.

It is possible to load the XML, save it in several way – like compound document or separated documents. User can use XSLT to transform it or use the scripting support to do arbitrary operation.

SumatraTT also offers colourful visualisation of the XML documents.

Support for Module Development

It is expected for larger use of SumatraTT there will be necessary to create new modules for specific tasks. Although it is possible to make arbitrary transformation using scripting, it will be much more convenient to use specialised modules.

SumatraTT provides wide support for fast module design. The only requirement for module is to implement the Processing class and an XML description of the module.

Moreover, there are multiple classes included in SumatraTT in order to provide a base for modules, so the module developer can concentrate only on the task. For most modules, it is sufficient to override the appropriate method(s) to include the desired behaviour.

A simple module can be implemented in few minutes by experienced user with Java language knowledge. In the world of open source software, using already developed modules as source examples is recommended and might be useful for new users.

For an example the Benchmark module has only three lines of code. The rest five lines are used for better visual formatting. All the metadata, commands and reading of data is left on the base class.

The graphical interface to the modules is the easiest one – it consists of simply a JFrame, the common Java SWING class, which can be designed in any graphical interface for Java (e.g. Netbeans). SumatraTT provides only storage for the parameters of modules.

Author of a new module should write documentation for its module in HTML format. This file is not mandatory, but it is strongly recommended. This documentation included in module's jar file is displayed in SumatraTT from mouse popup menu of the module.

Finally, the author can add the new module to the main Ant script, which automatically packs the modules and deploy jar files in the right directory.

6.2.6 Amount of Processed Data

One of the planned use of SumatraTT is to load data warehouses. For this purpose it has to be fast enough. Amount of such data is usually about hundreds of megabytes.

Currently implemented modules emphasise rather rich features than execution speed. For many data mining applications the amount of data is small and the effectiveness of the transformation run is not important comparing to number of available features in order to change the data in many different ways.

Nevertheless, the I/O speed is satisfying – load from comma separated files load approximately 40.000 lines per second⁴. The speed will further increase after the rewriting of the module using the Java New IO with low-level access to file.

⁴The measurement was performed on Pentim IV, 2.8 GHz, one record consisted of 30 attributes.

A similar situation is with the database connection, which is using advanced batch processing to speed up the database (un)load. Speed of access to database is rather given by the speed of connection to the database, effectivity of the particular database engine and its driver.

6.3 SumatraTT and Data Transformation Tasks

Present offer of data transformation modules provided by SumatraTT is determined by the needs of the data mining projects in which SumatraTT has been applied. We are not going to describe all currently available modules in detail. Instead we will review the basic groups of supported tasks by presenting them in several clusters, which are reviewed in the sequence they often appear during a DM task:

Transition between different data representations and formats. Data exchange between specified formats (e.g., from relational database to a set of Prolog facts) represents the simplest task in this cluster. Propositionalisation of a relational database represents much more complex problem belonging into this cluster – we are working on one possible solution to this problem [53], but the corresponding module is not implemented, yet. It should be noted that propositionalisation of some data sources can result in introduction of sparse attributes and data reduction has to be applied.

Data understanding. The aim is to produce a data survey providing basic information about the treated data as well as evaluation of their quality. One of the modules which fall into this cluster is the First Touch Review described in 6.3.1. Elementary visualisation techniques simplify presentation of the generated results to the user – see figure 6.7 for a practical example.

Handling missing values, extremes and errors in data. This set of tasks is most often ensured using statistical methods and it covers data cleaning and outlier detection as well.

Change of data volume. Modules in this cluster cover such tasks as adding background knowledge by enrichment (complementing information from other sources) or enhancement (deriving additional attributes from the original source data). Important example of enhancement is *reverse pivoting*, which is ensured by a dedicated module used often when handling time series data. Further, there are contained here the modules concerned with problems of data volume reduction. This can be achieved using various types of aggregation, principal component analysis or sparsely connected autoassociative neural nets. Handling sparse attributes belongs into this cluster as well.

Data visualisation has been introduced as an indispensable tool for communication with the SumatraTT user as well as with the domain experts [23]. That is why the corresponding means are treated in more details in the next subsection (see section 6.3.1).

Creation of data sources for modeling and evaluation. For modeling, the set of available data has to be divided into training and testing subsets. If the size of the original dataset is too big, the user can indicate the relative sizes of both parts and the corresponding *random division (sampling)* is applied. This division has to ensure that the statistical characteristics of the training set and of the original set remain unchanged. Other

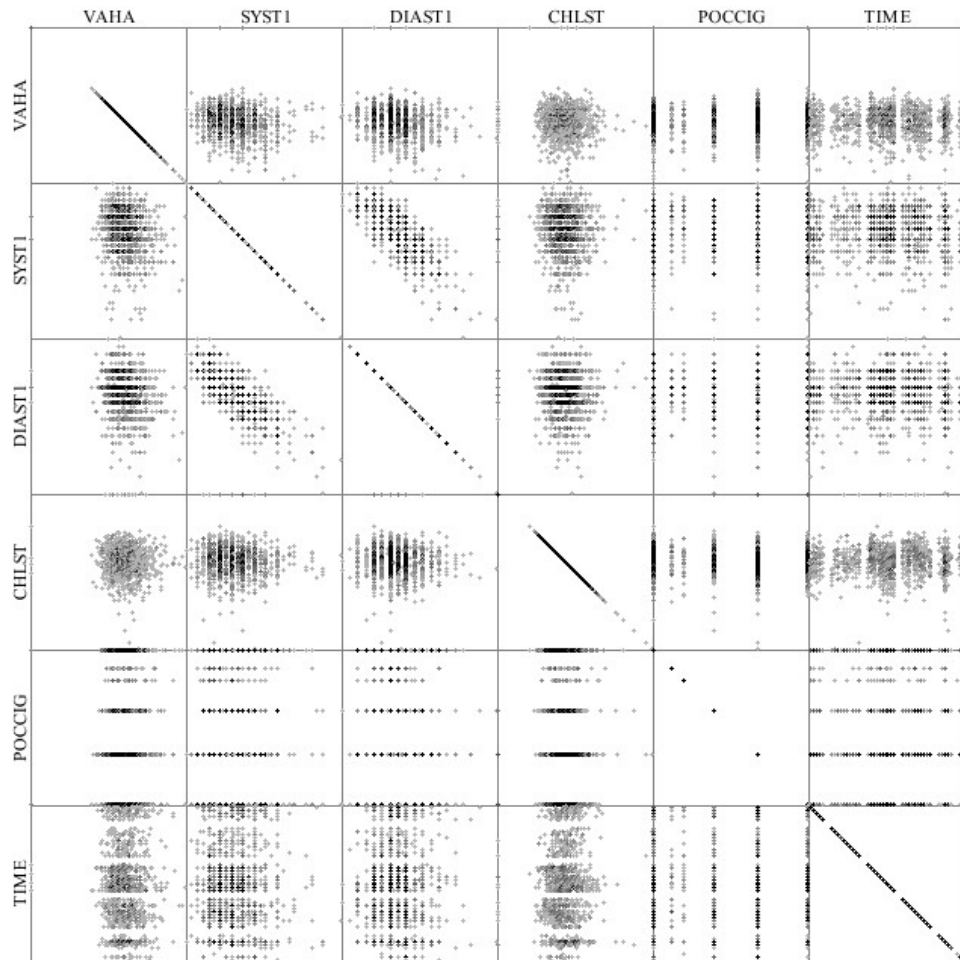


Figure 6.7: Scatter plot matrix for STULONG data – dependencies between pairs of selected attributes: weight, systolic pressure, diastolic pressure, cholesterol, nicotine level and time.

sophisticated approaches have to be used, when the source dataset is too small or biased.

6.3.1 Visualisation

As noted in the introduction, after loading data the data mining process continues by visualisation of the data as the first step in data understanding. In SumatraTT there are these levels of visualisation with increasing data understanding:

1. first-touch review
2. static
3. interactive
4. advanced

Each of these sets is already populated by a number of modules.

First Touch Review

Any data mining application commences by a report about the studied data from the point of view of each used attribute: the structure, distribution, and frequency of values has to be analysed separately for each attribute. Such a report is generated by the First-Touch Review which informs about elementary statistical characteristics of individual attributes and includes corresponding graphs and histograms. It serves as a quick overview over the analysed data and provides the user with input he/she will need to make the decisions concerning design of further processing, visualisation or choice of attributes which could be completely omitted (due to constant values, excessively erroneous values, etc.). *First-Touch Review* is implemented by a module which needs no setup.

Static

The static visualisation depicts relations between pairs of attributes and the corresponding output is generated without user's intervention (or with a very simple setup). Let us recall as typical examples of static visualisations e.g. tabular list of values, graphs (e.g. 2D/3D bars – see figure 6.8 or pie graphs – see figure 6.7), histograms, etc.

Interactive

Through interactive visualisation the user can reach more profound understanding of the role of the individual attributes and explore some dependencies between them by asking various questions. The data miner can e.g. click on a point in one of the 2D graphs and the system answers by a number of items in the source data which fall into this point. This goal is not achievable by static visualisation; hence an interactive environment is required. SumatraTT provides several kinds of extended versions of static forms, e.g. histograms and 3D graphs.

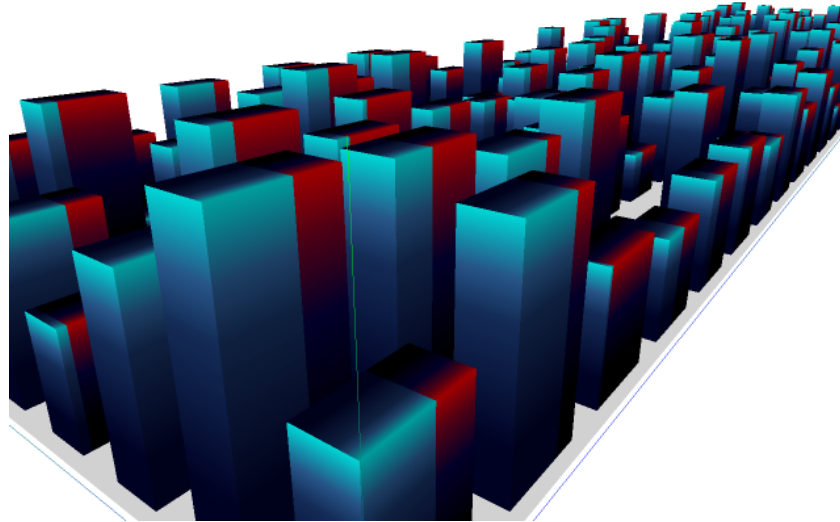


Figure 6.8: 3D graph with column split wrt. classification

Advanced

To understand the results of the last set of advanced visualisation techniques special knowledge is required. The user has to know what the particular type of the graph is able to depict and how the resulting image should be interpreted. Modules for RadViz visualisation or Parallel coordinates are typical representatives of this set.

6.3.2 Some additional modules

The openness and modularity of SumatraTT enable smooth cooperation between the visualisation and processing modules – due to this feature the transformation processes can be controlled using visual appearance of data.

A natural part of modules is its documentation, which can be displayed from its context menu. It is necessary for more complicated modules like scripting module or modules from advanced visualisation.

6.3.3 Practical Application in Data Preprocessing

SumatraTT has been used in several projects for data preprocessing.

In the GOAL project [31], it preprocessed data from several sources and transported data between data warehouse and geographical information system.

In the Sol-Eu-Net project, whole package dedicated to data preprocessing for data mining was solved by further developing of SumatraTT. For example, SumatraTT divided dataset with vehicle accident into separate datasets according to year. Experiences

with the project lead into a book [34] with a chapter describing data preprocessing and SumatraTT.

In the STULONG project [26], SumatraTT was used for a complex data preprocessing comprising advantage methods. The implementation of the used method enriched the set of available modules.

6.4 SumatraTT in Knowledge Management

6.4.1 Testing Modules

During a work on ontology processing in SumatraTT several modules have been prepared to test capabilities of both SumatraTT core and ontology engines. These modules mostly generated graphical output representing the ontology. As their source the modules use the ApolloCH library (see section 4.9.1).

Apollo2dot

The Apollo2dot was used for visualisation of a class inheritance. The module shows classes and the structure of the inheritance and instances. It generates source file for the GraphViz suite (particularly for dot). Its result is in figure 6.9 and shows a part of annotation of a stories for South Bohemia in the CIPHER project. Instances are represented by gray boxes.

PATExporter

One of the tasks of the CIPHER project was a generation of HTML variants of ontologies. Together with so called *Picture Annotation Tool* (PAT), annotating a spatial information by drawing and annotating regions on images, it was possible to generate a whole web presentation.

A sample of PATExporter output is shown in figures 6.10 to 6.12. It is a part of a bigger application guiding user through a set of historical monuments with information about it and links to related items.

DAML2Apollo

In the early stages of work on ontology transformations, the DAML2Apollo module allowed to investigate problems of migration of ontologies between the DAML and Apollo formalisms. The problems found led to further development of the whole GOF framework.

A development of this module ceased as it has been replaced by the GOF modules.

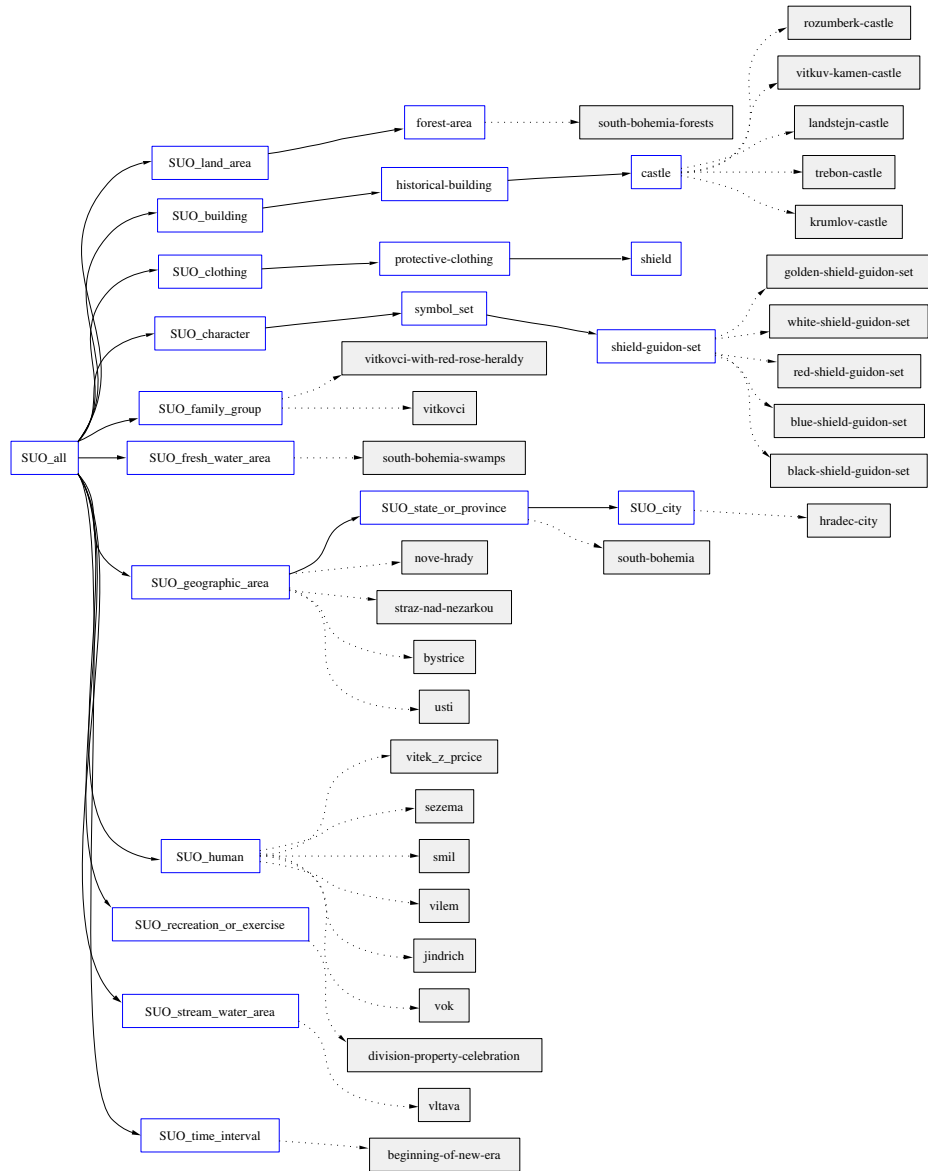


Figure 6.9: A tree of class inheritance generated by the Apollo2dot module

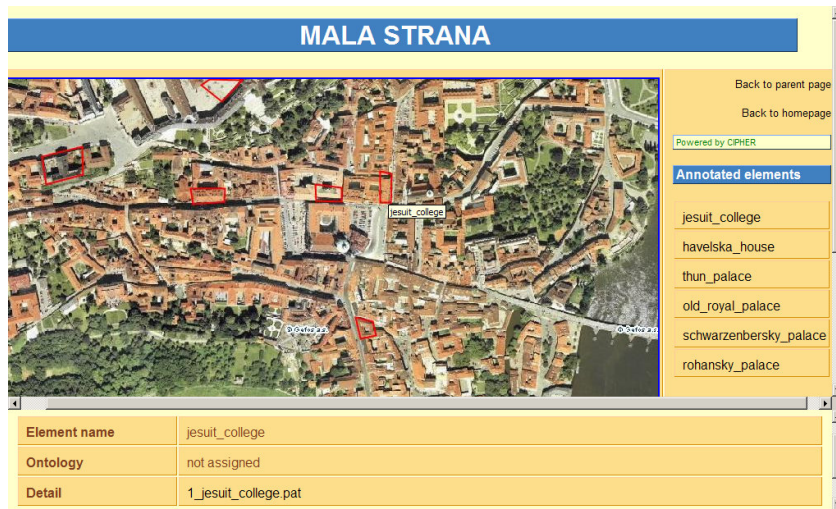


Figure 6.10: An entry page of the PATExporter output

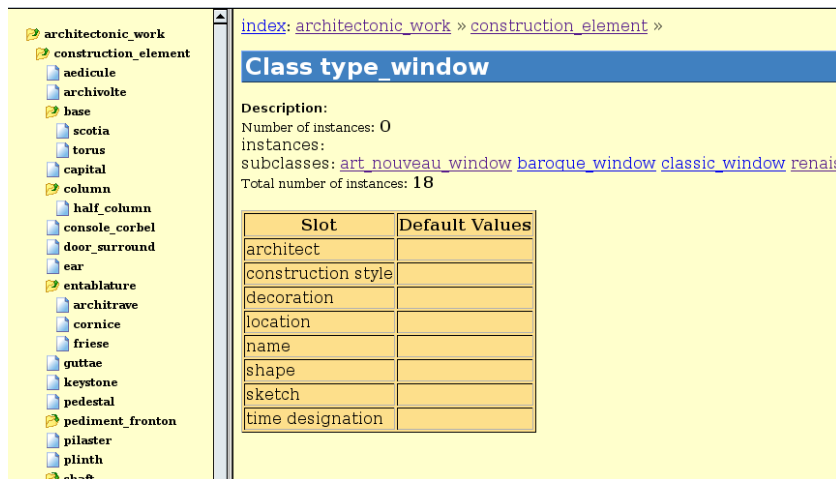


Figure 6.11: HTML pages generated by the PATExporter module with an index of classes

- 📁 architectonic_work
- 📁 construction_element
- 📁 palace
 - 📁 art_nouveau_palace
 - 📁 baroque_palace
 - 📁 classic_palace
 - 📁 renaissance_palace
 - 📁 renaissance_revival_palace
- 📁 kind_of_decoration
- 📁 shape_of_door_frame
- 📁 shape_of_roof
- 📁 shape_of_window_frame
- 📁 type_of_structure

CodeThat.Com

[index: architectonic_work](#) »
 [palace](#) »
 [renaissance_palace](#) »

Instance Schwarzenberg_palace

Description:

Sketch:



Slot	Values
architect	Agostino Galli

Figure 6.12: An instance with an image attached with a list of properties

6.5 Implementation of Generalised Ontology Formalism in SumatraTT

6.5.1 GOF Implementation

There are two usages of the GOF (see 5.6) in SumatraTT. First, the formalism library is used as a base for 22 modules representing gates and operations. Second, the GOF formalism can be used as a background knowledge for the interface about the processed data and can guide user in the selection of modules and their combination.

The SumatraTT kernel is flexible enough to support problem-specific data formats. Thus the support for ontology processing required no modification of the kernel. A set of modules was prepared to process GOF in SumatraTT.

Processing GOF in SumatraTT helps users to use the framework in a graphical way and conveniently set up parameters.

6.5.2 Module Conversion

As soon as the GOF framework had been tested and produced a reasonable output, a need to transform GOF gates into SumatraTT modules emerged. Instead of manual creation of the modules, the process was made automatised.

A translation program has been developed making SumatraTT modules from GOF gates – *Moduliser*. Each gate is asked for support of loading/saving and a corresponding SumatraTT module(s) is (are) generated together with a simple GUI form.

If the gate supports loading of ontology in the given formalism, *Moduliser* creates a module called `FromXXX`, where `XXX` is the name of the gate. The same happens for saving, the name is then `ToXXX`.

All the new modules are then copied to the directory, where all SumatraTT modules reside, and compiled as a part of the whole SumatraTT suite.

6.5.3 Example

In figure 6.13, there is a screenshot of ontology processing in the SumatraTT system. The example shows loading two ontologies from the Apollo formalism, making union and diff and saving the union back to Apollo and diff to both GraphViz (graphical tool) and HTML table.

A migration of ontology between formalisms is rather easy to perform – the whole design consists of two interconnected modules as shown in figure 6.14. A more complex structure is done, when a verification step is required after the migration. An example of a transformation with verification is in figure 6.15.

6.5.4 Background Knowledge of Transformation

SumatraTT is not only a means for ontology transformations. It also offers an interesting application of knowledge management techniques. There is a chance to build an onto-

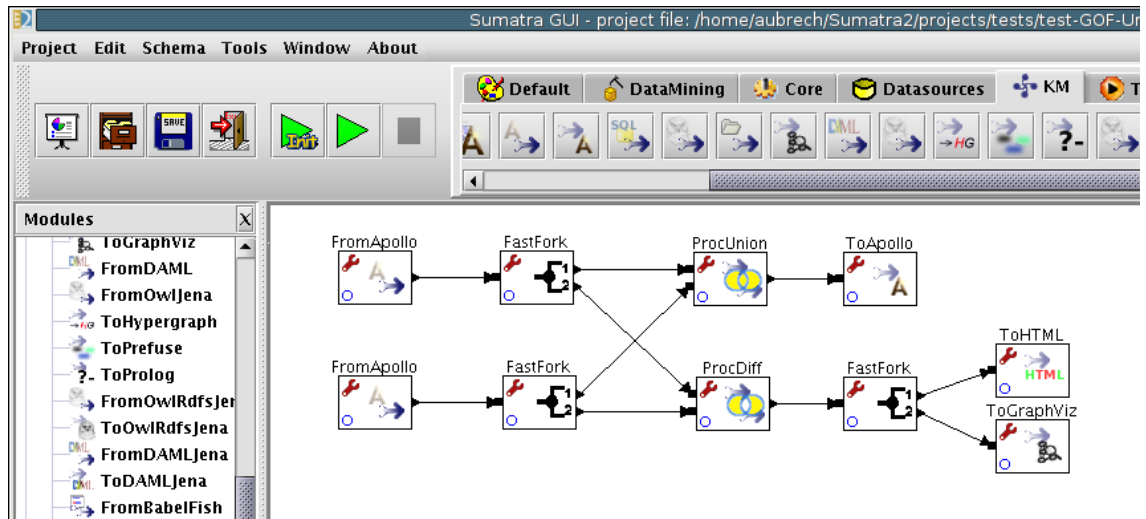


Figure 6.13: Ontology processing in SumatraTT

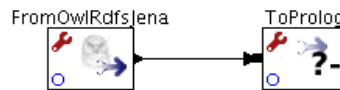


Figure 6.14: An ontology migration between the OWL and Prolog formalisms

logy allowing description of successfully reusable transformations – see Best Patterns in section 6.2.2.

6.6 SumatraTT Summary

GOF provides a base framework for research of ontology formalism conversion. This formalism is used as a dictionary in the process of information migration between different systems for both data conversion and information interchange between the systems.

The relative simplicity of GOF makes it appropriate for developing transformations to/from many formalisms and sharing ontologies from multiple sources in a uniform way. It also makes easier performing operations on ontologies based on either equal or different formalisms such as determining differences between ontologies, merging ontologies etc.

The generalised ontology formalism is implemented as a part of a system SumatraTT, which provides a platform for generic data transformations. GOF is supported with SumatraTT modules aimed at loading and storing the model from/to different ontology formalisms (automatically generated from GOF gates) and for various kinds of data transformation and visualisation.

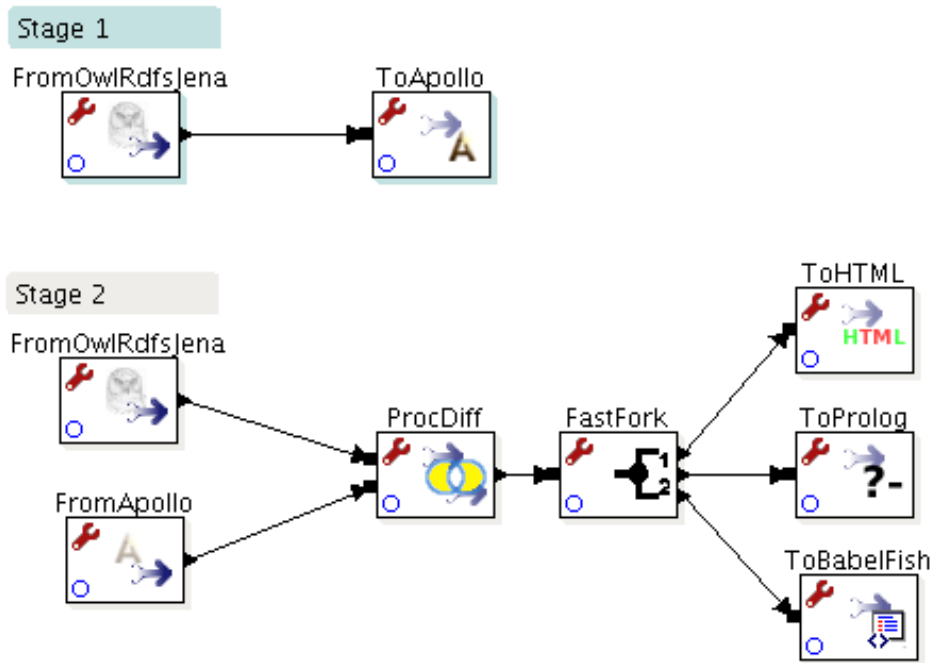


Figure 6.15: An ontology migration with verification

7 Conclusion

This thesis presents a new ontology formalism for ontology sharing, called Generalised Ontology Formalism (GOF), consisting of six relations between concepts. Its purpose is supporting migration of ontologies between formalisms with as small as possible information loss. Among others, GOF provides a means for defining meta-models of formalisms and thus it makes possible to study and compare their expressive power.

The theoretical part of the thesis introduces definitions of *ontology*, *ontology grammar*, and *formalism*. These definitions allow distinguishing between the set of expressing constructs (ontology grammar), which make possible to encode ontology, and the set of all ontologies utterable by means of this set of expressing constructs (formalism). Such a theoretical basis provides means for an explicit separation of the structural part of ontology from the procedural one. All common ontology formalisms have been described by means of this theoretical framework.

A framework for converting the structural part of ontology between different formalisms has been designed, implemented, and verified. The main idea consists in expressing the ontology by means of GOF. Thus, the ontology is transformed between respective formalisms in two steps – first from the source formalism into GOF and then to the target formalism. This allows migration of the structural part between any supported formalisms.

The transformation have been analysed from the point of view of information loss during the conversion. The conditions for achieving lossless transformation were stated. Two different methods of ontology transformations – called *informed* and *non-informed* ones – were proposed.

For a pair of formalisms, between which the migration is expected to be carried out frequently, specific (informed) transformation can be defined. The informed transformation makes use of the knowledge of both the source and the target formalism meta-models. Thus, a set of transformation rules, which is specific for the given pair of formalisms, can be prepared and utilised in the course of the particular conversion. Because of its specificity, the informed transformation shall provide better accuracy of ontology conversion than the non-informed one, which does not use the knowledge of mapping between the source and target formalism meta-models.

The non-informed transformation makes possible to quickly include a new formalism into the framework. The reason is that only $2n$ transformations are necessary for migrating between any two of n formalisms. On the other hand, $2\binom{n}{2}$ transformations are necessary in the case of the informed one.

Particular formalisms are processed by so called *gates*, which transform ontologies from the particular formalism to GOF and vice-versa. For each gate, the respective Formalism-Specific Ontology (FSO) has been developed in terms of GOF, which defines

7 Conclusion

the set of meta-level concepts used by the formalism (concepts like *Class*, *Instance*, etc.).

It has been shown, that GOF can handle all the structures, which occur in all common ontology formalisms, using various combinations of GOF relations. An advantage of this simple formalism is the ability to ignore relations, which are not recognised by a particular gate, i.e. they have no corresponding analogy in the respective formalism. In this way, ontologies can migrate between formalisms of very different expressive capabilities without a need of a purposely written converter.

The generic data processing system SumatraTT, which was designed by the author of this thesis few years ago, has been chosen as the implementation basis for the theoretical framework introduced by this thesis. SumatraTT has been equipped with an algorithm making possible to automatically generate SumatraTT modules from descriptions of respective gates. The respective gates' FSOs are included for purposes of the informed transformation.

The whole framework was verified on a number of ontologies of various size including SUMO and Cyc, which is the largest publicly available ontology.

GOF is planned also as a platform for annotating an archive of the *Best Patterns* of using SumatraTT modules. Thus, SumatraTT is not only a means for implementing knowledge management applications, but it becomes an application domain itself.

Bibliography

- [1] Petr Aubrecht and Luboš Král. Ontology Formalism Transformation. In Fernando Galindo, Makoto Takizawa, and Roland Traummüller, editors, *Database and Expert Systems Applications – DEXA 2004*, pages 95–99. IEEE Computer Society, September 2004.
- [2] Petr Aubrecht, Petr Mikšovský, and Luboš Král. SumatraTT: a Generic Data Pre-processing System. In *Database and Expert Systems Applications*, pages 120–124, Heidelberg, 2003. Springer. [5.6, 6](#)
- [3] Petr Aubrecht and Monika Žáková. Ontology Formalism Transformation. In Karel Ježek, editor, *DATAKON 2004*, pages 191–200. Masaryk University in Brno, 2004. [5.6.4](#)
- [4] Petr Aubrecht, Monika Žáková, and Zdeněk Kouba. Ontology Transformation Using Generalised Formalism. In *Znalosti 2005 – sborník příspěvků 4. ročníku konference*, pages p. 154–161. VŠB-TUO, 2005. (in Czech).
- [5] Franz Baader and Philipp Hanschke. A Scheme for Integrating Concrete Domains into Concept Languages. Technical Report RR-91-10, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH, Erwin-Schrödinger Strasse, Postfach 2080, 67608 Kaiserslautern, Germany, 1991. [4.6](#)
- [6] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, May 2001. [3.6](#)
- [7] Pim Borst. *Construction of Engineering Ontologies for Knowledge Sharing and Reuse*. PhD thesis, Tweente University, 1997. [4.2](#)
- [8] Hans Chalupsky. OntoMorph: A Translation System for Symbolic Knowledge. In *Principles of Knowledge Representation and Reasoning*, pages 471–482, 2000. [5.3.1](#)
- [9] Pete Chapman, Julian Clinton, Randy Kerber, Thomas Khabaza, Thomas Reinartz, Colin Shearer, and Rüdiger Wirth. *CRISP-DM 1.0: Step-by-step data mining guide*. CRISP-DM consortium, 2000.
- [10] Oscar Corcho and Asunción Gómez-Pérez. A Roadmap to Ontology Specification Languages. In *Proceedings of the 12th European Workshop on Knowledge Acquisition, Modeling and Management*, pages 80–96. Springer-Verlag, 2000.

BIBLIOGRAPHY

- [11] Czech Technical University in Prague. SumatraTT Official Homepage, 2004. krizik.felk.cvut.cz/sumatra. 6
- [12] RNDr. Jiří Demel. *Grafy a jejich aplikace*. Academia, Akademie věd České republiky, Legerova 61, 120 00 Praha 2, 2002. 5.6.5
- [13] Marie Duží and Pavel Materna. Konceptuální modelování a ontologie z pohledu logiky. In Karel Ježek, editor, *DATAKON 2004*, pages 99–118. Masaryk University in Brno, 2004. 5.6.12
- [14] Jerome Euzenat and Heiner Stuckenschmidt. Family of Languages' Approach to Semantic Interoperability, 2001. 5.3.4
- [15] Dieter Fensel, Ian Horrocks, Frank van Harmelen, Stefan Decker, Michael Erdmann, and Michel C. A. Klein. OIL in a Nutshell. In *EKAW '00: Proceedings of the 12th European Workshop on Knowledge Acquisition, Modeling and Management*, pages 1–16. Springer-Verlag, 2000. 4.7.7
- [16] Thomas R. Gruber. A Translation Approach to Portable Ontology Specifications. *Knowl. Acquis.*, 5(2):199–220, 1993. 4.2, 4.4, 4.5.3
- [17] Thomas R. Gruber. Towards Principles for the Design of Ontologies Used for Knowledge Sharing. In N. Guarino and R. Poli, editors, *Formal Ontology in Conceptual Analysis and Knowledge Representation*, Deventer, The Netherlands, 1993. Kluwer Academic Publishers. 4.3.2, 5.1
- [18] Ladislav Hejdaňek. *Nepředmětnost v myšlení a ve skutečnosti*. Oikúmené. OIKOY-MENH, Prague, Czech Republic, 1997.
- [19] Ian Horrocks. Knowledge Acquisition ontology , 2003. <http://www.cs.man.ac.uk/~horrocks/OWL/Ontologies/ka.owl>. 5.6.13
- [20] Ian Horrocks, Peter Patel-Schneider, and Frank van Harmelen. From SHIQ and RDF to OWL: The Making of a Web Ontology Language. *Journal of Web Semantics*, 1(1):7–26, 2003. 4.6
- [21] Ian Horrocks and Ulrike Sattler. Ontology Reasoning in the SHOQ(D) Description Logic. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, 2001. 4.6
- [22] ICOM/CIDOC. Definition of the CIDOC Conceptual Reference Model, Version 3.4.10, March 2004. http://cidoc.ics.forth.gr/docs/cidoc_crm_version_3.4.10.pdf. 4.10.5
- [23] Michal Jakob. Symbolic Rule Extraction and Visualization using Network Function Inversion. pages 230–231, Technická 2, 166 27 Prague 6, 2004. In Workshop 2004 [CD-ROM]. 6.3

- [24] Aditya Kalyanpur, Daniel Pastor, Steve Battle, and Julian Padget. Automatic Mapping of OWL Ontologies into Java. In *Proceedings of Software Engg. - Knowledge Engg. (SEKE) 2004*, Banff, Canada, June 2004. 4.5.6
- [25] Jörg Uwe Kietz, Anca Vaduva, and Regina Zücker. MiningMart: Metadata-Driven Preprocessing. In *Proceedings of the ECML/PKDD Workshop on Database Support for KDD*, 2001.
- [26] Jiří Kléma, Lenka Nováková, and Olga Štěpánková. Trend Analysis in Stulong Data. In *Proceedings of the Discovery Challenge 2004 – A Collaborative Effort in Knowledge Discovery from Databases*, pages 56–67, Prague: University of Economics, 2004. 6.3.3
- [27] Vladimír Mařík, Olga Štěpánková, and Jiří Lažanský a kol. *Umělá inteligence 3*. ACADEMIA,, 2002. 4.3.2
- [28] Kamil Matoušek. *Representation and Processing of Uncertain Historical Time Periods*. PhD thesis, Czech Technical University in Prague, Faculty of Electrical Engineering, Technická 2, 166 27 Prague 6, Czech Republic, 2004. 4.5.4, 5.1, 5.6.8
- [29] Michael R. Genesereth. Knowledge Interchange Format (KIF), 1998. <http://logic.stanford.edu/kif/kif.html>. 4.4, 4.5.2
- [30] Petr Mikšovský and Zdeněk Kouba. GOLAP – Geographical Online Analytical Processing. In *Proceedings of the 12th International Conference on Database and Expert Systems Applications*, pages 442–449. Springer-Verlag, 2001. 6.1
- [31] Petr Mikšovský and Zdeněk Kouba. Application A2 Specification. Technical report TR11, INCO–COPERNICUS 977091 GOAL, Czech Technical University, Department of Cybernetics, Technická 2, 166 27 Prague 6, 1999. 6.3.3
- [32] George Miller. The Magical Number Seven, Plus or Minus Two. *Psychological Review*, 63:81–97, 1956. 5.6.14
- [33] Marvin Minsky. A Framework for Representing Knowledge. *The Psychology of Computer Vision*, McGraw-Hill, 1975. 4.1, 4.5
- [34] Dunja Mladenic, Nada Lavrac, Marko Bohanec, and Steve Moyle, editors. *Data Mining and Decision Support: Integration and Collaboration*. Kluwer Academic Publishers, 2003. isbn 1-4020-7388-7. 6.3.3
- [35] Katharina Morik and Martin Scholz. The MiningMart Approach to Knowledge Discovery in Databases. In Ning Zhong and Jiming Liu, editors, *In Handbook of Intelligent IT*. IOS Press, 2003.
- [36] Enrico Motta. *Reusable Components for Knowledge Modelling: Case Studies in Parametric Design Problem Solving*. IOS Press, 1999. 4.4

BIBLIOGRAPHY

- [37] Paul Mulholland, Trevor Collins, and Zdenek Zdrahal. Story Fountain: Intelligent Support for Story Research and Exploration. In *Proceedings of the ACM Conference on Intelligent User Interfaces (IUI'2004)*, 2004. 5.1
- [38] Lenka Nováková, Jiří Klema, and Olga Štěpánková. Anachronistic Attributes and Data Mining. In *MIPRO 2004 - Computers in Education*, pages 153–156, Croatia: Mipro HU, 2004.
- [39] Natalya F. Noy, Michael Sintek, Stefan Decker, Monica Crubezy, Ray W. Ferguson, and Mark A. Musen. Creating Semantic Web Contents with Protégé-2000. *IEEE Intelligent Systems*, 2(16):60–71, 2001. 4.9.1
- [40] Marek Obitko. Ontologies Description and Applications. Technical Report GL–126/01, Czech Technical University, Department of Cybernetics, Technická 2, 166 27 Prague 6, 2001. 4.4
- [41] Marek Obitko, Václav Snášel, and Jan Šmíd. Ontology Design with Formal Concept Analysis. In Václav Sášel and Radim Belohlávek, editors, *CLA*, volume 110 of *CEUR Workshop Proceedings*, 2004. 4.3.3
- [42] Dorian Pyle. *Data Preparation for Data Mining*. Morgan Kaufmann Publishers, Inc., San Francisco, CA, USA, 1999. 6.1.2
- [43] Xiaolei Qian. Correct Schema Transformations. In *Extending Database Technology*, pages 114–128, 1996.
- [44] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2003. 4.10
- [45] John F. Sowa. *Knowledge Representation: Logical, Philosophical and Computational Foundations*. Brooks/Cole Publishing Co., 2000. 4.1, 4.2, 4.10
- [46] John F. Sowa. Architectures for Intelligent Systems. *Special Issue on Artificial Intelligence of the IBM Systems Journal*, 41(3):331–349, 2002. 4.1
- [47] Stanford Medical Informatics. Protégé Project Homepage, 2004. <http://protege.stanford.edu/>. 5.4
- [48] Olga Štěpánková, Jiří Klema, Štěpán Lauryn, Petr Mikšovský, and Lenka Nováková. Data Mining for Resource Allocation: A Case Study. In *5th IEEE/IFIP Int. Conf. on Information Technology for Balanced Automation Systems (BASYS 2002)*, pages 477–484, Cancún, Mexico, 2002. Kluwer Academic Publishers.
- [49] Heiner Stuckenschmidt. Ontology-Based Information Sharing in Weakly Structured Environments, 2002. 5.3, 5.3.3
- [50] Vojtěch Svátek. Ontologie a WWW. In D. Chlapek, editor, *DATAKON 2002*. Brno : Masaryk University, 2002. 4.4

- [51] Alan M. Turing. Can a Machine Think? *Mind*, 59(236):433–460, 1950. 3.2
- [52] Monika Žáková. Semantic Annotations. Master’s thesis, The Czech Technical University in Prague, 2005.
- [53] Filip Železný. Efficiency-conscious Propositionalization for Relational Learning. *Kybernetika*, 4:275–292, 2004. 6.3
- [54] Regina Zücker and Jörg Uwe Kietz. How to preprocess large databases. In *In Data Mining, Decision Support, Meta-learning and ILP: Forum for Practical Problem Presentation and Prospective Solutions*, 2000.