Die approbierte Originalversion dieser Diplom-/Masterarbeit ist an der Hauptbibliothek der Technischen Universität Wien aufgestellt (http://www.ub.tuwien.ac.at).

The approved original version of this diploma or master thesis is available at the main library of the Vienna University of Technology (http://www.ub.tuwien.ac.at/englweb/).

FAKULTÄT FÜR !NFORMATIK

Interaktive Visualisierung Semantischer Graphen

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Master of Science

im Rahmen des Studiums

Computergrafik und Digitale Bildverarbeitung

eingereicht von

Stefan Müller
Matrikelnummer 0426886

am:
Institut für Computergraphik und Algorithmen

Betreuung:
Herr Ao. Univ.-Prof. M. E. Gröller
Herr Ao. Univ.-Prof. Dr. Keith Andrews

Wien, _____ _____ _____

Unterschrift Verfasser Unterschrift Betreuer

Kurzfassung

Wissensbasierte Systeme (WBS) stellen für viele Anwendungen einen immer wichtigeren Bereich dar. Aus Anwendersicht erlauben es WBS neues Wissen aus der Wissensbasis zu folgern und bestehende Daten effizient zu verwalten. Aus Entwicklersicht wird die schnelle Anpassung an einen Problembereich ermöglicht, da Wissen von Geschäfts-Logik getrennt wird. Das m2n Intelligence Management Framework stellt eine Möglichkeit dar, WBS, die komplett durch einen semantischen Graphen beschrieben sind, im Rapid-Prototyping-Verfahren zu erstellen. Dieser Graph wird mit Hilfes des Resource Description Framework (RDF) definiert.

In dieser Arbeit werden Möglichkeiten aufgezeigt den semantischen Graphen einer m2n Anwendung zu visualisieren und interaktiv zu manipulieren. Dazu wurden existierende Werkzeuge zur Visualisierung von RDF- und OWL-Daten sowie Technologien aus dem Bereich der Informations-Visualisierung untersucht. Darauf aufbauend wurde die bestehende Visualisierung überarbeitet und Techniken zur direkten Manipulation, zur Fokus & Kontext Visualisierung und zum Linking & Brushing integriert.

Die entwickelte Komponente kann auf beliebige Anwendungsfälle, wie z.B. die Exploration und Modellierung von Ontologie-, Geschäftsprozess- und Instanz-Daten, angepasst werden. Um die Entwicklung zu evaluieren, wurde ein Thinking Aloud Test mit acht Nutzern des Systems durchgeführt. Die Ergebnisse des Tests fließen in die Weiterentwicklung der Komponente ein.

Schlüsselwörter:

Semantische Graphen, Visualisierung multivariater Daten, Interaktive Modellierung, Linking & Brushing, Focus & Context, Excentric Labels, Preattentive Wahrnehmung, RDF, OWL, Usabilty, Thinking Aloud.

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Ort:	Wien
Datum:	_____
Unterschrift:	_____

Interactive Visualization of Semantic Graphs

MASTER THESIS

Abstract

Knowledge-based systems (KBS) are playing an increasingly important role in real-world applications. Such systems allow their users to manage huge amounts of information efficiently and to make inferences from that data. For developers, the use of KBS allows new applications to be rapidly built, since the knowledge modeling and reasoning processes are cleanly separated. The m2n Intelligence Management Framework supports the creation of KBS in a rapid prototyping approach. The resulting applications are more "modeled" than "programmed" and are completely described by an RDF-based semantic graph.

This thesis explores the possibilities for the interactive visualization and manipulation of the semantic graph describing a m2n application. Both existing applications for visualizing RDF and OWL data and techniques from the field of information visualization were investigated. Based on the findings, the existing m2n graph visualization component was redesigned and enhanced through a variety of focus & context, linking & brushing, and direct manipulation methods.

Formative evaluation was conducted in the form of a thinking aloud test with eight users and the results were used to further improve the user interface. The new GraphVisualization component can be adapted to arbitrary use cases, such as the exploration and modeling of an ontology, business workflows, or instance data.

Keywords:

semantic graphs, multivariate data visualization, interactive modeling, linking & brushing, focus & context, excentric labels, preattentive perception, RDF, OWL, usabilty, thinking aloud.

Acknowledgment

I am indebted to my colleagues at m2n who finally gave me the time to finish this work and allowed me to try out different approaches even if they did not make it into the final version of this thesis. I would also like to thank my supervisor Keith Andrews for his help with correcting the draft versions of this work, providing feedback, and giving inspiration. The same applies to my supervisor Eduart Gröller. This work is dedicated to Olivia, who gave me the energy to finish this work.

Stefan Müller
Vienna, Austria, March 2009

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis presents the ongoing research and development of the m2n GraphVisualization Plug-In (GVP). The GVP is part of the m2n Intelligence Management Framework (IMF) which is used to build knowledge-based systems (KBS). The GVP visualizes facets of the knowledge base which is represented as a semantic graph. When this work was started, the GVP had only a weak support for visual browsing and manipulation methods. Usability problems with the old component motivated the investigation of visualization and interaction techniques to enhance the semantic graph visualization. Moreover the most common use cases were analyzed to provide example user scenarios for a formative evaluation. The results of these investigations are presented in the following chapters.

The next chapter, Chapter 2, describes the foundations of semantic graphs and their roots in cognitive science, in logic, and in philosophy. Semantic graphs have commonalities and can be seen as distinct approaches for the same problem.

Chapter 3 gives an overview of the m2n Intelligence Management Framework and explains its basic components including MultiTree, DetailView and GraphVisualization. Furthermore, the chapter describes the basic concepts related to the m2n Framework which will be referred to in the remainder of this thesis. Beyond that, a more detailed motivation for this work is given.

Chapter 4 reviews common visualization techniques from the field of information visualization. Attention is paid to focus+context visualization and the display of multivariate network data. Moreover, techniques to direct attention in visualizations are discussed which make the representation effective even if the screen is cluttered with information. In Chapter 5, a review of existing tools examines which techniques are actually used and how they could contribute to the improvement of GraphVisualization.

Chapter 6 describes the requirements for the new GraphVisualization Plug-In and how these requirements were collected. Furthermore, the chapter describes a user model, two common usage scenarios and a simple conceptual model which serves as a framework for later extensions. The user model explains the target groups of GVP and describes two hypothetical users. The two scenarios described serve to define the common use cases.

Chapter 7 and 8 describe the features of the implemented framework component. Chapter 7 elucidates the features from a user's point of view. Chapter 8 details how the visualization was implemented on top of JGraph and the m2n GUI Plug-In. Beyond that, some implementation obstacles are explained.

The redesigned GraphVisualization Plug-In was evaluated using a thinking aloud test. A group of eight test users, matching one of the identified target groups, took part in the test. The exact methodology and the results are explained in Chapter 9.

Chapter 2

Semantic Technologies

In 1970 Ed Feigenbaum, Bruce Buchanan, and Joshua Lederberg invented a system called DENDRAL (Feigenbaum et al. [1970]; Lederberg [1987]). The program, they announced, was able to do the work of an analytical chemist. It took the mass and nuclear resonance spectra of an unknown chemical compound and produced a visual representation of the molecular formula. The revolutionary part of their approach was the ability to derive new answers from declared expert knowledge about chemical compounds. It was possible to add new compounds without changing the derivation process, since the data representation had been decoupled from the inference mechanism. This idea became fundamental in the development of so-called *expert systems* and the more sophisticated *knowledge-based systems* (Bobrow et al. [1986]; Russel and Norvig [2004]).

Knowledge-based systems (KBS) are playing an important role in applications today (Hayes-Roth and Jacobstein [1994]). For a user, the KBS supports the management of huge amounts of data. Moreover, new data can be derived through reasoning mechanisms. For a developer, the separation of declared knowledge and deduction allows faster adaption of the system to new problems and simplifies the maintenance of large scale applications.

This chapter explains the base technologies and their application within the m2n Intelligence Management Framework. The next section describes the components of KBS in more detail. Section 2.2 defines the term *semantic graphs* and outlines their general properties. The term *ontology* is introduced and explained in more detail. Finally, the *Resource Description Framework* (RDF) and the *Web Ontology Language* (OWL) are described, since the m2n Framework is built on top of them.

2.1 Knowledge-Based Systems

Knowledge-based systems (KBS) usually contain two important parts ([Russel and Norvig, 2004, p. 250–252]). A *knowledge base* (KB) which stores information about the world and an *intelligent agent*. The agent is able to query and manipulate the information stored in the KB. Moreover, the agent processes the declared knowledge by making inferences about it. Abstractly, an intelligent agent can be described by a logic system whith three parts (Kreuzer and Kühling [2006]; Franconi [1996]):

Symbols Symbols are also denoted as atomic formulas, or primitives. They can be combined using conjunctions and operations following a specific *syntax* to create logic *statements*. Symbols, conjunctions and operations provide the *vocabulary* of the system. Logic systems can be divided by their *ontological commitment*. That is, the entities that are regarded as symbols. For example, propositional logic only regards facts as symbols, whereas first order logic, which is important for KBS, regards facts, objects and relations as symbols.

Semantics Semantics assign meaning to logic statements. The semantics define a set of rules which map logic statements to truth values. For example, in propositional logic the value of $a \vee b$ is mapped to false, only if a and b are false (non exclusive or). This mapping is also called the *epistemological commitment* which expresses the state of knowledge or belief in a statement.

Calculus A calculus defines mechanical transformation rules for logic statements, which can be used to prove the satisfiability of a logic statement.

A crucial task when developing KBS is the identification and representation of the information to be stored in the knowledge base. Any task related to this process is part of the *knowledge engineering* (Reimer [1998]). The information gathered during this process is denoted as *domain knowledge*. Knowledge engineering also includes the choice of the logic system and its representation language.

2.2 Semantic Graphs

Semantic graphs, also called semantic networks, provide the means to represent knowledge in a way similar to the human memory model (Sowa [1992]). The description of a semantic network can rely on a logic system or can be done intuitively by humans. A semantic graph may also provide a defined visual representation such that visual artifacts can be mapped to logic symbols. The purpose of such visualization is to allow a human to interpret and manipulate the graph easily.

2.2.1 Assertional Networks

Assertional networks are a kind of semantic graph used to make assertions about the world. These assertions are known, or believed to be true (Sowa [1992]). The idea of these graphs is rooted in cognitive science and in logic.

In cognitive science, semantic graphs are founded on observations of how humans learn about their environment (Canas et al. [2005]; Novak and Canas [2008]). When humans perceive regularities about objects or events in their environment, they start to organize, sort and, relate things. Therefore, they apply names to perceived *concepts* and their *relations*. The resulting network of information can be represented as a directed graph. Therein, the nodes form the concepts and edges represent statements, assumptions or beliefs about these concepts. Figure 2.1 shows an example of an intuitively defined graph called *concept map*. The map organizes concepts related to the gas "ozone" and can easily be interpreted by a human. For example, it is possible to read that ozone occurs in the troposphere and the stratosphere of the earth.

In logic, assertional semantic networks are based on the work of Charles Sanders Pierce, who sought a way to represent logic statements as a node-link diagram (Sowa [1992]). Sowa [1979] extended his approach of *existential graphs* and defined so-called *conceptual graphs*. Figure 2.2 shows an example of a conceptual graph that expresses the sentence "Maggy hit the piggybank with a hammer". Conceptual graphs define two types of nodes. The first type, conceptual nodes, represent a *generic concept* or an *individual* of a generic concept. For example, the node PERSON:Maggy is an individual of the generic concept PERSON. Concepts appear as boxes. The second type of node represents the relation between two concepts and is called a *conceptual relation*. A conceptual relation has at least two arcs which connect concepts or individuals. For example, the node AGNT (agent) is a conceptual relation that defines a dependency between the individual concept Person:Maggy and the concept HIT. Conceptual relations appear as circles in the graph.

The main difference between conceptual graphs and the intuitively defined concept maps is their clearly defined semantics. For example, the node PERSON:Maggy refers to a logic statement that Maggy is an element of the set PERSON. The set PERSON can be further described by other concept graphs. The same applies to the conceptual relation HIT. The relation is defined as a combination of individuals

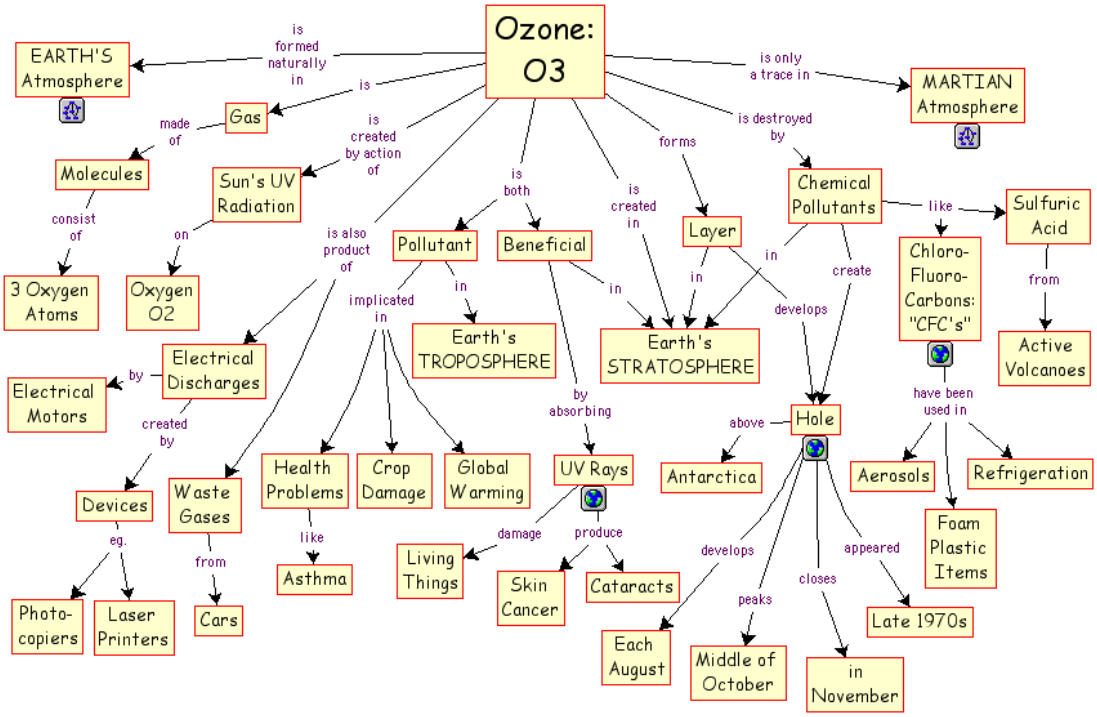


Figure 2.1: An example of a concept map describing knowledge about ozone. Background knowledge is required to understand this kind of semantic graph. Image extracted from Novak and Canas [2008]. Used under the terms of the Austrian Copyright Law (UrhG) §46.

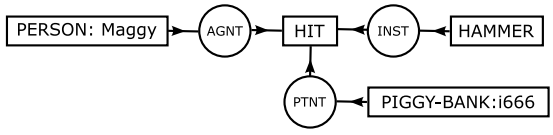


Figure 2.2: Example of a conceptual graph expressing the sentence "Maggy hit the piggy bank with the hammer". The relational nodes AGNT (agent), INST (institution) and PTNT (patient) define the relationship between the concepts Hammer, HIT, and the individual concepts PERSON:Maggy and PIGGY-BANK:i666. Image extracted from Sowa [1979]. Used under the terms of the Austrian Copyright Law (UrhG) §46.

from PERSON, HAMMER and PIGGY-BANK. In other words it can only occur if individuals for these three concepts exist.

In contrast, background knowledge is required to interpret the concept map in Figure 2.1. For example, the nodes "Earth's STRATOSPHERE" and "Earth's TROPOSPHERE" refer to a place (earth) and a part of earth's atmosphere. The node "Living Things" is another example. A user could think of plants, animals, and humans when asking what is damaged by UV rays. However, the set of living things damaged is not precisely defined. For a knowledge-based system, the concepts and their relations must be explicitly defined to support the reasoning process. This means concepts must be mapped to symbols which can be combined into logic statements having clear semantics. A possibility to achieve this is given by definitional networks.

2.2.2 Definitional Networks – Ontology

In its original sense the term ontology refers to a philosophical discipline that seeks the truth about real or ideal things and their interrelations with each other. Philosophers use the word "metaphysics" as a synonym to indicate that they do not only refer to real things. The goal of this discipline is to find a definitive and exhaustive classification of all things in the universe (Smith [2004]).

In computer science, the term is used in a more restricted sense and is mostly limited to the description of the concepts in a certain field. An often cited definition is that of Tom Gruber, who refers to an ontology as an explicit conceptualization of a specification (Gruber [1993]). This means, an ontology describes the concepts in terms of a vocabulary or schema. Thus, an ontology provides the ontological commitment for a particular *domain*.

There are multiple ways to provide such a vocabulary. The first programs to explicate their knowledge used *dictionaries*. For example, in 1969 M. Ros Quillian presented a program called the "Teachable Language Comprehender" (Quillian [1969]). The program analyzed English sentences and linked their words to definitions in a dictionary. These links could be used to access further information about the objects in a sentence. This made the program appear to actually understand sentences.

In 1974 Marvin Minsky proposed a construct called a *frame* to describe knowledge (Minsky [1974]). Minsky tried to understand how humans could understand a three-dimensional scene, even if it looks different from every point of view. He concluded that there must be a more generalized structure of the scene that a human can remember. A frame is similar to the concepts introduced earlier, since it bundles a set of properties that are common to the different viewpoints. Thus, a frame can be thought of as a template or *generic concept*. Minsky states that a frame has two types of properties (slots). First, it can be associated to other frames. The association can represent a generalization or specialization such that frames can be organized in a hierarchy. The more general frames appear in the top of the hierarchy and more specialized frames appear at the lower levels. More specialized frames also inherit the slots of the more general frames. The second kind of slots refers to *primitive concepts* like numbers or names. Minsky called these slots *terminal slots* and assumed that a generic frame would have meaningful default values which can be overridden.

An example of a language which implements the frame approach is the Knowledge Language One (KLONE) (Schmolze and Lipkis [1985]). Figure 2.3 shows an excerpt of an ontology that uses the graphical notation of KLONE to define the concept of a truck and a trailer truck. The white nodes refer to the generic frames. The node "Integer" has a star beside it to mark it as a primitive concept. The gray node containing the label "18" is an *individual concept* of this primitive concept. Moreover, there are two kinds of roles (slots) in this graph. The double-line arrows represent the hierarchy of the concepts and denote a subclass/superclass relationship. Thus, a truck is a special kind of vehicle. The relation `CargoCapacity` is an example for the second kind of role since it defines a property of a generic concept. The abbreviation "r/v" beside the arrowhead makes clear, that the range of the property is restricted to concepts of type VolumeMeasure.

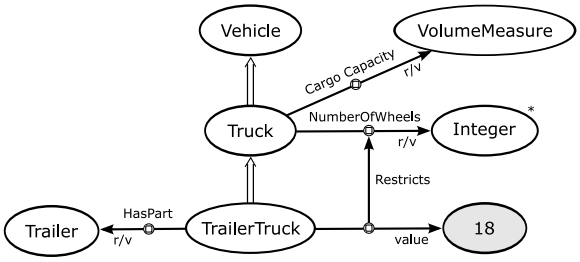


Figure 2.3: An excerpt of a vehicle ontology in the graphic notation of KLONE. Image adapted from Sowa [1992]).

2.2.3 Upper Level Ontology

Ontology, as used in computer science, can be seen as the opposite to that in philosophy. In philosophy the domain is identical with the whole universe while in computer science the term is restricted to a small domain. In other words, philosophy applies a top down view to the domain while computer science uses a bottom up approach. The second approach is practical for small worlds but it suffers from generality when applied to another domain. In other words, such an ontology can not be shared easily since the constraints which hold in one domain need not be valid in another (Guarino and Poli [1995]; Guarino [1998]; Smith et al. [2005]).

To overcome this problem, a hierarchy of such small world descriptions must be built using shared concepts. A conceptualization used in different domains must agree on the concepts in higher level ontologies. The Basic Formal Ontology (BFO) is a well-known advocate of this approach that should serve as a root of ontologies (`http://www.ifomis.org/bfo`). In general, it should be kept in mind that vocabularies can be organized in a hierarchy.

2.2.4 General Components of Semantic Graphs

Semantic graphs provide the means to represent knowledge as a directed graph with different visual representations. For KBS, the representation must use clearly defined semantics, such that the elements can be mapped to logic statements. The following terms will be used interchangeably throughout this thesis:

Concept/Class/Frame The terms (generic) concept, class, and frame are used interchangeably. A concept is denoted with a label and is mostly represented as a single node or as a set of nodes. Formally, a class can be represented as a set of *individuals* or *objects*. The class determines which attributes and roles its individuals can or must have. For example, in class diagrams in the Unified Modeling Language (UML), each individual must have the properties defined for that class.

Individual/Instance/Object An object is an instance of a class. That means it is a member of the set defined by a class. Formally an object is itself a set of *attributes* and *roles*.

Attribute/Terminal Slot and Primitive/Literal Attributes are also denoted as *terminal slots* and denote properties like `age` or `name`. These properties are similar in a way as they refer to actual values.

Primitive Type/Literal Terminal slots refer to actual values with a datatype like integer or string. The value of a terminal slot is therefore denoted as primitive type or literal.

Slot/Role/Property Roles define relations between objects or classes. In other words, they do not refer to primitive types. Often, two types of relationships are provided. First, the *`is a`* role. The `is a` role means that an individual belongs to a class and therefore must or can have the properties defined for that class. Formally, the `is a` relation means that an object belongs to the set which is

defined by a class. The second property is the `subclass of` relationship to define specializations. This relation is also denoted as *subsumption* (Nebel and Smolka [1992]). Formally, if a class B is a subclass of class A, then $B \subset A$.

2.3 Semantic Graph Representation

As shown in the examples above, there are multiple ways to describe semantic graphs visually. However, to store the information in a knowledge base, a well defined representation format is required. Since the semantic web can be seen as a semantic graph that spans the entire internet, semantic graph description languages like the Resource Description Framework (RDF), RDF Schema (RDFS), and the Web Ontology Language (OWL) turn out to be appropriate to represent knowledge in a KBS (Lassila and McGuinness [2001]; Lee [1999]).

The following sections give a short introduction to the basic concepts of the RDF, RDFS, and OWL. A good starting point about RDF is the RDF-Primer (Manola and Miller [2004]) and the RDF concepts description given in Klyne et al. [2004]. The semantics of RDF is defined in Hayes and McBride [2004]. The RDF Schema language definition and its relationship to RDF is described in Brickley and Guha [2004]. Finally, the OWL description relies on McGuinness and Harmelen [2004] and Smith et al. [2004]. The following description is based on these documents.

2.3.1 Resource Description Framework (RDF)

The original purpose of the RDF was to add metadata to web resources by making statements about them. A statement is an assertion about the world that is assumed to be true. A set of RDF statements form an RDF graph which is an assertional network as described in Section 2.2. Each statement in such a graph consists of a *subject*, a *predicate*, and an *object*. Thereby, each of these elements can be identified by a Unique Resource Identifier (URI). As in natural language, the subject refers to the entity (resource) being described. The predicate is a resource which associates a *property* with the subject and can be thought of as a role. The object refers to the value of the property and can be a resource or a *literal*. A literal is a leaf in the graph which actually represents a value such as a number, a string, or a date. Due to these three elements, statements are also denoted as *triples*.

RDF only defines the data model for semantic graphs based on triples, but it does not restrict the developer to a single syntax. In this work, all examples will be given in Notation 3 (N3) as introduced in Lee et al. [2003]. To exemplify this, consider the statements in Listing 2.1 that describe the RDF graph shown in Figure 2.4. The listing defines statements declaring knowledge about living things and ultraviolet rays previously shown in the concept map in Figure 2.1. Each statement is a set of three words referring to the subject, the predicate, and the object. A statement is finished by a dot at the end of the line. If the subject should be shared with multiple predicates, the first statement can be terminated by semicolon as shown in line 5–7 of Listing 2.1. Lines starting with a hash (#) are comments and do not contribute to the graph definition. In a statement, a resource is usually written as a full URI or is abbreviated using the schema `prefix:element`. The prefix (namespace) defines the scope of `element` and must be defined in advance using the `@prefix` declaration. However, to keep listings short, the declaration is omitted in all other examples in this work. Instead, all namespaces are declared in Appendix A.

As shown in Listing 2.1 and Figure 2.4, the graph contains information about the resources "Living Thing" and "UV Rays". Line 5 declares that a resource ex:UVRay has a relationship with the resource ex:LivingThing. Lines 9–11 define that the resources ex:Human, ex:Animal and ex:Plant have the type ex:LivingThing. Based on these statements, the RDF semantics define how these resources are also affected by the `ex:damage` property. The same applies for the resources ex:Peter, ex:Jumbo and ex:Rose.

```
@prefix rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns# .
@prefix ex:  http://www.example.org/ .
@prefix xsd: http://www.w3.org/2001/XMLSchema# .

ex:UVRay ex:damage ex:LivingThing .
ex:UVRay ex:maxWavelength "380"^^xsd:integer ; # note the semicolon
         ex:minWavelength "1"^^xsd:integer .

ex:Human  rdf:type  ex:LivingThing .
ex:Animal a         ex:LivingThing . # rdf:type is equal to 'a'
ex:Plant  a         ex:LivingThing .

ex:Peter a ex:Human .
ex:Jumbo a ex:Animal .
ex:Rose  a ex:Plant .

# valid triples but nonsensical
ex:Rose ex:damages  ex:UVRay ;
        ex:eat      ex:Jumbo .
```

Listing 2.1: Example RDF Graph to declare knowledge about living things and ultraviolet rays. Figure 2.4 shows a graphic representation of these statements.

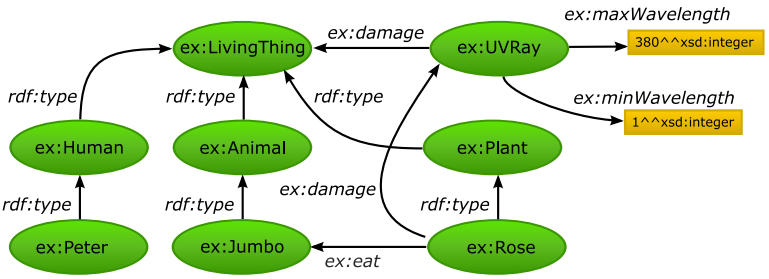


Figure 2.4: Example of an RDF graph visualization using the graphic notation introduced in Manola and Miller [2004]. Green nodes represent resources, yellow boxes show literals, and dark edges represent the statements defined in Listing 2.1.

It should be noticed that RDF does not impose any meaning on the names used in the statements. Taken literally, the last two statements are valid triples but do not make sense semantically. A rose can not damage ultraviolet rays and can not eat some animal named jumbo. However, in terms of RDF these triples are valid statements since they could be true. Furthermore, there is no assumption that ex:Human is a class and ex:Peter is an individual. In fact, the RDF vocabulary does not provide the means to declare something to be a class. RDF itself only provides the means to assert something through statements whereby the same URI refers to the same entitiy. It is up to the interpreter to apply a meaning to the predicates, subjects, and objects.

2.3.2 RDF Schema (RDFS)

RDFS is a vocabulary description language which supports the definition of a frame and slot-based ontology. Therefore, it provides resources in a special namespace that can be used to describe classes, properties, and their relations. This way, RDFS supports the creation of a type hierarchy that is similar to that of object-oriented programming languages. However, the approach differs from the common programming paradigms. First of all, RDFS defines a top level element of the vocabulary which is called

```
1  ex:LivingThing a rdfs:Class .
2
3  ex:Human  rdfs:subClassOf ex:LivingThing .
4  ex:Animal rdfs:subClassOf ex:LivingThing .
5  ex:Plant  rdfs:subClassOf ex:LivingThing .
6
7  # restrict the domain and range of the "damage" predicate
8  ex:damage a rdfs:Property ;
9      rdfs:domain ex:UVRay ;
10     rdfs:range  ex:LivingThing .
```

Listing 2.2: RDFS statements which extend Listing 2.1 to define a more useful vocabulary.

rdfs:Resource. Any resource in the vocabulary is an instance of this class. Secondly, RDFS defines a resource to describe classes called rdfs:Class. To declare a resource as a class, the property `rdf:type` must be added to that class as shown in Listing 2.2, line 1. To define subclasses, RDFS provides a property called rdfs:subClassOf. This is shown in lines 3–5 of Listing 2.2. As mentioned above, this defines a containment of sets. However, in RDFS it is also possible to define infinite loops of containment since a class can be a subclass and even an instance of itself.

Using the subclass property it is possible to define taxonomies. To describe classes or to apply restrictions on properties, two further elements are required. First, the property `rdfs:Domain` can be used to restrict the subjects of a predicate. Second, the property `rdfs:Range` can be used to restrict the objects of a predicate. To apply these restrictions, the properties are added to the *reification* of a predicate. This means, the resource of the predicate appears as subject in a statement and is further described. Listing 2.2 gives an example for the property `ex:damage`. In line 8 the predicate `ex:damage` is reified and declared as an instance of rdfs:Property. Lines 9–10 declare the domain and the range. Based on these statements, a knowledge base should not allow any longer to add the statement between ex:Rose and ex:UVRay shown in line 18–19 of Listing 2.1 Figure 2.4.

2.3.3 Web Ontology Language (OWL)

The Web Ontology Language (OWL) is another possibility to define a vocabulary for a particular domain. In comparison to RDFS, OWL has a higher expressiveness. In essence it means that OWL defines an additional set of resources and properties having predefined semantics for an inference system. Three increasingly expressive variants of OWL exist: OWL Lite, OWL DL and OWL Full.

The OWL Lite vocabulary provides a migration path for simpler vocabularies like thesauri and RDFS taxonomies. A feature introduced by OWL Lite is the distinction between *datatype properties* and *object properties*. Object properties always refer to resources while datatype properties refer to primitives. Another feature introduced is the ability to mark properties as the inverse of another. For example, it would be possible to define a property `ex:damagedBy` that is the inverse of `ex:damage`. The inference mechanism must then ensure that if property `ex:damage` is defined for two individuals, the inverse must be defined es well.

The OWL DL and OWL Full vocabulary add further expressiveness whereby OWL DL is guaranteed to be computable in finite time. A feature added by these vocabularies are arbitrary cardinality constraints for properties. While OWL Lite supports cardinalities to be 0 or 1, OWL DL allows the defintion of any positive number. Thus, it would be possible to restrict the number of wavelength statements in Listing 2.1 by adding a property `owl:cardinality`, `owl:minCardinality` or `owl:maxCardinality`. However, the example has another problem that could only be solved by OWL DL. RDFS has no means for expressing that two classes are disjoint. Thus, it would be possible to declare an individual as an animal, a plant and

a human at once. In OWL DL this could be prohibited by adding a `owl:disjointClass` to the classes of animal, human, and plant.

OWL defines a much larger set of properties that can not be mentioned in detail here. The purpose of the examples given above is to show how a vocabulary could be defined for a KBS by the means of RDF. Furthermore, the terms *datatype property* and *object property* will be used throughout this work.

2.4 Summary

This chapter gives an introduction to semantic technologies starting with a description of knowledge-based systems and semantic graphs. Semantic graphs provide the means to declare the knowledge for a KBS, if their semantics are defined in terms of a logic. Several examples were given whereby the most recent technologies, RDF, RDFS, and OWL play the most important role within this work. It had been outlined how these technologies are related to each other and which features they provide to express knowledge. The common visual representation relies on a simple node-link diagram which quickly becomes cumbersome. The second issue with RDF-based graphs is the lack of proper modeling tools. It is easy to forget a semicolon or a dot in the N3 definitions above. Therefore, an easier way to manipulate and declare RDF data is required. This work addresses how RDF data can be visualized and manipulated in a certain context with respect to the needs of a KBS user.

Chapter 3

The m2n Intelligence Management Framework

The m2n Intelligence Management Framework (IMF), developed by the company m2n, serves as a base for the creation of knowledge-based systems (KBS). The basic architecture relies on the IMF server and IMF client. The server manages a global knowledge base stored by means of RDF. The client provides the user interface to query and manipulate the knowledge base. This program is a Java tool which is based on Swing, or a Java Servlet that is based on WingS. The functionality of client and server is further divided into Plug-Ins. Functionality is only available if a corresponding Plug-In is available.

An important feature of the framework is that the knowledge base not only contains domain knowledge, but also describes all other aspects of the client. This includes the appearance and the functionality. Thus, any button shown in the user interface of the client is a resource which can be identified by an URI. The same applies to the function triggered by a button. The resulting semantic graph is called the *application graph* or *working model*. In this sense, a natural way of thinking about an m2n application is to think of it as an RDF interpreter. When changing the triples in the knowledge base, the resulting program is changed as well.

The framework uses an extended schema of OWL, which is referred to as OWL Extended (OWLX). This schema defines several base classes used to configure an application. For instance, the class owlx:InstantiableClass is defined. This resource is the base class for all resources which can be created by the user in its domain. Such resources have further properties like `owlx:className` and `owlx:classIcon`. The class name defines a human readable string appearing in the user interface. The predicate `owlx:classIcon` can be used to assign a symbolic icon. To distinguish between domain knowledge and application data, OWLX defines another important class called owlx:Zielontologie. Any resource that is important for the problem domain of a customer should be a subclass or instance of this resource. The graph defined by these subclasses and their relations is denoted as *target ontology* in the following text. The creation of the target ontology is one of the main tasks when building a client for a customer.

3.1 m2n GUI Plug-In

The GUI Plug-In describes the user interface of the client by the means of RDF. Fundamentally, this approach is comparable to other user interface definition languages. For instance, the XML User interface Language (XUL) used in the Mozilla projects also defines the structure of the user interface. Additionally, it uses cascading style sheets to define the appearance of the interface. The major difference in the m2n approach results from the use of RDF. In RDF information can be added at any time and anywhere. This means the statements of the example given in Listing 3.1 do not necessarily reside in a single file,

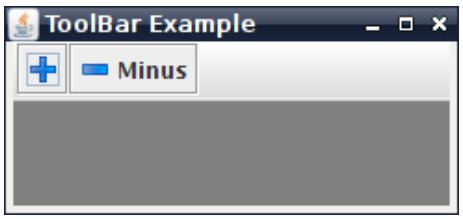


Figure 3.1: Example of a toolbar showing the Swing representation of the toolbar defined in Listing 3.1.

```
1  ex:Toolbar a    component:toolbar ;
2      toolbar:visible "true"^^xsd:boolean ;     # make the toolbar visible
3      toolbar:floatable "false"^^xsd:boolean ;  # make the toolbar not detachable
4      toolbar:elements [                        # define the contents...
5          a   rdf:seq ;
6          rdf:_1 ex:Button1;
7          rdf:_2 ex:Button2;
8          # to be continued ...
9      ] ;
10 .
11
12 ex:Button1 a   component:button ;
13     button:icon "plus.png"^^xsd:string ;    # button specific
14     vc:tooltip "Plus"^^xsd:string ;         # inherited property
15 .
16
17 ex:Button2  a   component:button ;
18     button:text "Minus"^^xsd:string ;
19     button:icon "minus.png"^^xsd:string ;   # button specific
20 .
```

Listing 3.1: RDF description of a toolbar containing two simple buttons. The first button shows only an icon and provides a tooltip. The second button shows a label but no tooltip. The namespace declaration is omitted for convenience (See Appendix A for details).

but must be available in the knowledge base somewhere. Therefore, it is not required to use another language like cascading style sheets to decouple the design from the interface structure description.

Listing 3.1 defines the simple toolbar shown in Figure 3.1. The first line of Listing 3.1 defines a new resource of type `component:toolbar`. Internally, the framework maps the URI of this type to a Java class that either provides a toolbar for Swing or the web client. The same applies to the buttons defined by `ex:Button1` and `ex:Button2`. With respect to the RDF type, the framework evaluates further properties of the resources. For example, the property `button:icon` (line 13) is specific to buttons. Other properties, like `vc:tooltip`, are common to several visual components. In other words, the property `vc:tooltip` is inherited from the base class of all GUI components called VisualComponent.

The definition of a user interface itself is not sufficient to adapt a client for a specific application. It is also required to add the functionality through *rules*.

3.2 Rules and Workflows

A *rule* is an algorithm described by the means of RDF having a fixed number of *rule blocks*. Figure 3.5 shows an example visual representation of a rule containing *methods* (rectangular blocks), *variables* (round nodes) and *mappings* (spline edges). Methods can be thought of as functions in the terms of

```
1  # define a method block
2  ex:GetSelectedItem a block:MethodGetSelectedListItem ;
3      blockCallMethod:next    ex:IfSelectedItem .
4
5  # define a mapping
6  ex:mapping a mapping:Class ;
7      mappingClass:sourceBlock    createNewObject:GetSelectedItem ;
8      mappingClass:destBlock      createNewObject:IfSelectedItem ;
9      mappingClass:mapFrom        outparam:bIsResource ;
10     mappingClass:mapTo          ifparameter:condition .
11
12 # define a conditional block
13 ex:IfSelectedItem
14     a   block:If ;
15     blockIf:then    ex:SomeThenBlock ;
16     blockIf:else    ex:ShowElseBlock .
```

Listing 3.2: Example of a conditional block definition.

a programming language. Like the user interface components, the framework associates the URI of a method with an actual piece of code. Methods can also provide input and output parameters. To assign a value to a parameter, the framework uses mappings. A mapping is a class which provides properties to identify the target and the source block in a rule. In other words, mapping resources describe the data flow in a rule. The control flow is described by the rule blocks themselves. Every rule block provides object properties referring to the next rule block.

Listing 3.2 shows an example of a method block and the conditional block `ex:IfSelectedItem`. The first line defines a method block referring to some code which selects an item from a list. Line six of Listing 3.2 defines a mapping ex:mapping assigning the result of the method to the conditional block ex:IfSelectedItem. The conditional block evaluates this assignment on execution and maps the result to its respective next blocks. In this case, there are two possible next blocks: ex:SomeThenBlock and ex:SomeElseBlock.

3.3 Common Components

The user of an m2n IMF application will always be faced with three basic components: the DetailView, the MultiTree, and the GraphVisualization. These three components are subject to configuration when the framework is adapted for a particular application. Given the RDF description of the user interface, it is possible to change the functionality and the appearance at runtime by the means of rules. For example, the DetailView described in the following section is configured with respect to a certain resource. This resource is set by a rule when the DetailView is opened.

3.3.1 DetailView

The DetailView provides a frame and slot view of a selected resource. To provide a better overview, the object properties and datatype properties are distributed over tabs. The availability of these tabs partly depends on the resource being inspected. Each tab follows the same composition and has two columns: the first column contains the name of the property, the second column contains the value and usually an editor. The editor shown depends on the range of the respective property. For example, the range `xsd:date` leads to a date editor. For object properties, a simple list is shown containing the associated resources. These resources can be opened in a new DetailView by double-clicking. Figure 3.2 shows the

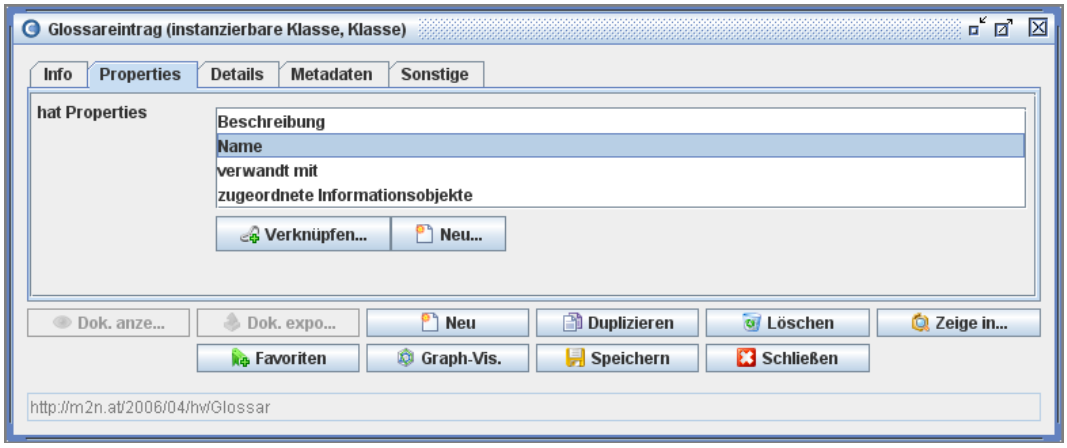


Figure 3.2: DetailView of a class that defines a glossary entry. The properties tab allows to create new resources or links to existing ones. Screen capture of m2n IMF 5.0.

properties tab of the DetailView for the class named Glossareintrag which shows the object properties and the datatype properties of the resource.

The DetailView also provides the means to create resources or links to existing resources. For this purpose, two buttons are shown under each row of object properties. The button "Neu" is used to create a new datatype or a new object property. The button "Verknüpfen..." is used to create a link (statement) to an existing resource. Pressing this button executes a query which returns all resources in the range of the property currently selected in the list above. The result of the query is visualized using a MultiTree component. Therein, the user can choose the resources to appear as the objects of the new statement.

The main problem of this method results from the huge amount of available resources. It is often required to scroll within a large tree which makes modeling tedious and error prone. Moreover, it takes time to open the MultiTree since all resources that fit the range must be collected from the working model. The same problem occurs for the creation of new properties, but is even worse. In this case, the user needs to deal with multiple instances of the DetailView where it is easy to loose the orientation.

3.3.2 MultiTree

The MultiTree queries the knowledge base and visualizes the results as a tree. Resources in the result set are grouped with respect to a configurable property. For instance, the tree shown in Figure 3.3 shows all resources in the target ontology. The columns of the tree can show additional information based on the properties of the resource. In the example, the type of the property is rendered in the second column.

Like the DetailView, the MultiTree can be used to create and link new resources. This is usually achieved through the context menu of the tree, which provides menu entries for the creation of properties and subclasses. In any case, creation opens a DetailView where the user can enter the required information. In other words, the modeling drawbacks of the DetailView also apply to the MultiTree.

3.3.3 Graph Visualization

The third common way to access the resources stored in the working model is the GraphVisualization. This component is the main subject of this thesis and will be further analyzed in Chapter 6. In m2n Intelligence Management Framework 3.5, the GraphVisualization is implemented as an extra Plug-In but not as a VisualComponent. This is worth mentioning since the configuration possibilities are limited for this reason.

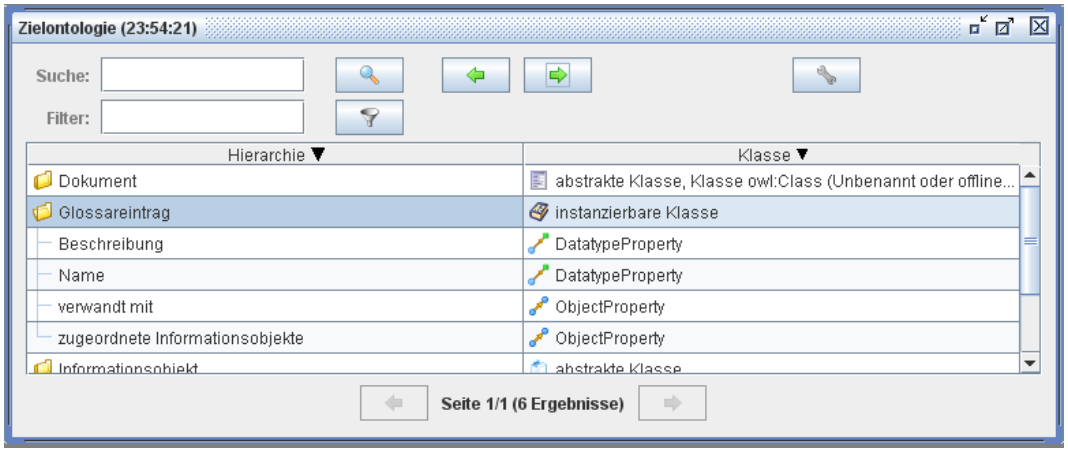


Figure 3.3: A MultiTree showing all subclasses of the target ontology including their object and datatype properties. Screen capture of m2n IMF 5.0.

The GraphVisualization component provides three different layouts for accessing the resources in the application graph. The first possibility is the StarVisualization (Figure 3.4). This visualization starts at a particular resource of the application graph and traces certain object properties. A a consequence, the nodes in this visualization represent classes, properties, or instances. Using the StarVisualization, it is possible to inspect the graph step by step. After double-clicking a node, the system collects the configured properties from the application graph and arranges the associated resources in a circle around the clicked node.

The second layout type, the RuleVisualization, is used to display rules containing the elements described in Section 3.2. The nodes display the different rule blocks, whereas edges display control and data flow. Figure 3.5 shows an example of a rule which allows users to add a bookmark to their personal profile.

The third common layout of the GraphVisualization is the exploration of the target ontology with the OntologyVisualization. When visualizing the target ontology, nodes display their direct subclasses. Each node shows the class name and a small part of the class description. If no name is set, the URI of the resource or the property is shown instead. The edges in this visualization show the object properties of the resources. Thereby, the property `owlx:directSubClass` is represented by an curved edge to achieve a visualization similar to class diagrams in UML. Straight edges are used for all other object properties of the resources. Figure 3.6 shows an example of the OntologyVisualization for the Upper Austrian Provincial Archive.

3.4 Motivation

The adaption of a client for a new application domain requires the creation of the target ontology, the user interface, and the rules of the client. A common procedure during the kickoff of a new project is to define the target ontology at meetings with the customer. The customer describes the knowledge domain and an engineer from m2n models the domain in the framework. The creation of rules and the adaption of the GUI requires more time and is the subject of later development phases. In both cases, the three main components, DetailView, MultiTree and GraphVisualization are used to build and adapt the underlying RDF constructs. Thereby, the StarVisualization version is mostly used for the exploration of the application graph.

One fundamental problem is that modeling rules and ontology graphs require many repetitive steps, which becomes a very tedious task. The modeler needs to deal with multiple instances of the DetailView and the MultiTree. The modeling support in the GraphVisualization is limited as well since any action

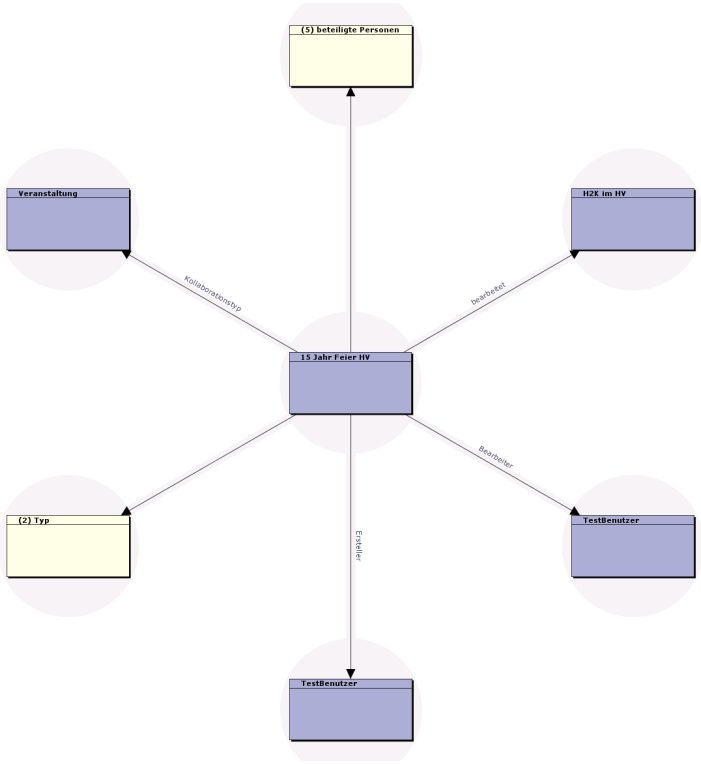


Figure 3.4: StarVisualization in the m2n Intelligence Management Framework 3.5

must be triggered from the context menu: direct manipulation of resources is not possible. Moreover, the exploration and handling of graphs containing more than 20 nodes becomes a problem due to the poor performance and the lack of orientation features.

The scope of this thesis is to improve the visualization in order to simplify the exploration and modeling process for developers using the m2n IMF. The three types of visualization discussed above serve as an example to define common tasks and requirements. Further specific types of visualization, like organizational charts or formula trees, might be useful in the future. Thus, any improvements should allow interaction with a broad spectrum of semantic graphs and users.

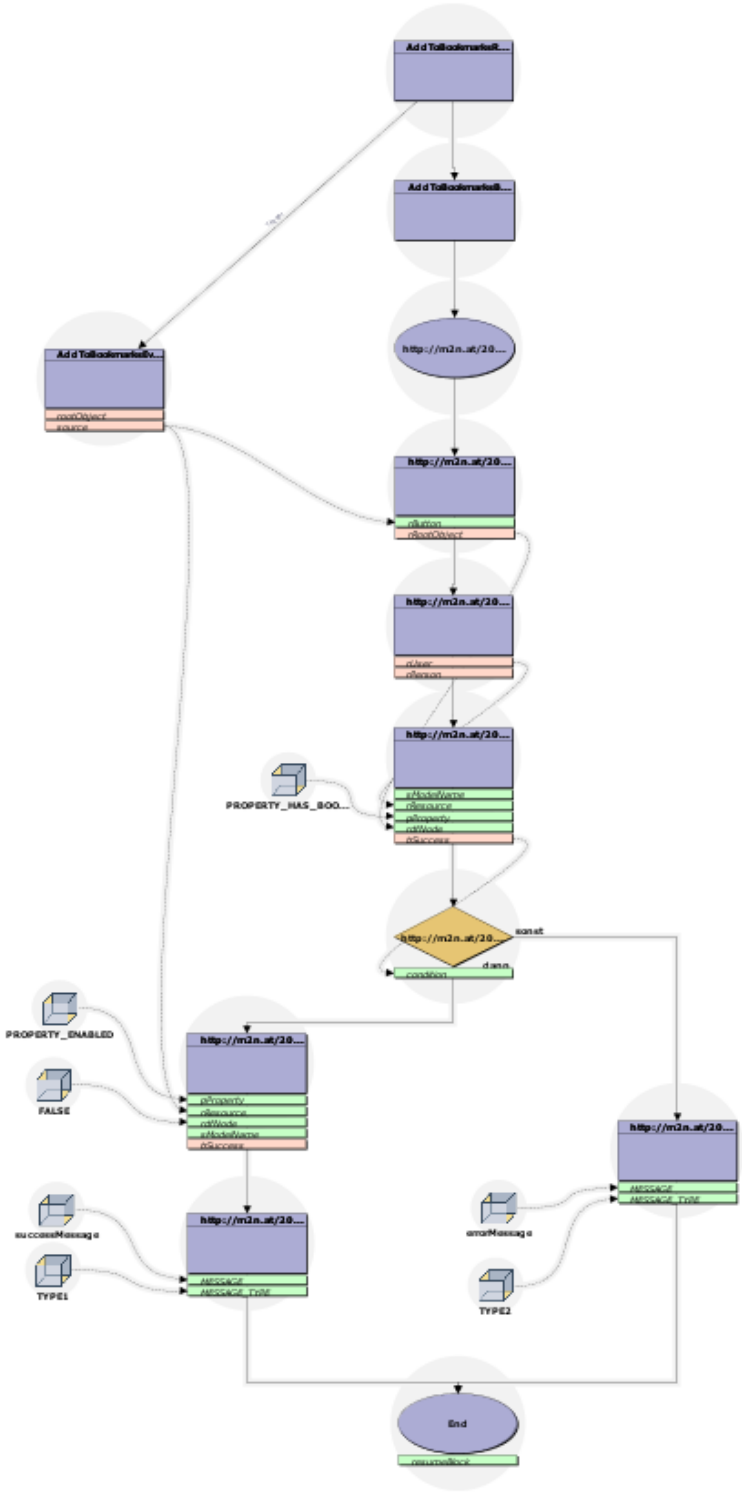


Figure 3.5: RuleVisualization in the m2n Intelligence Management Framework 3.5

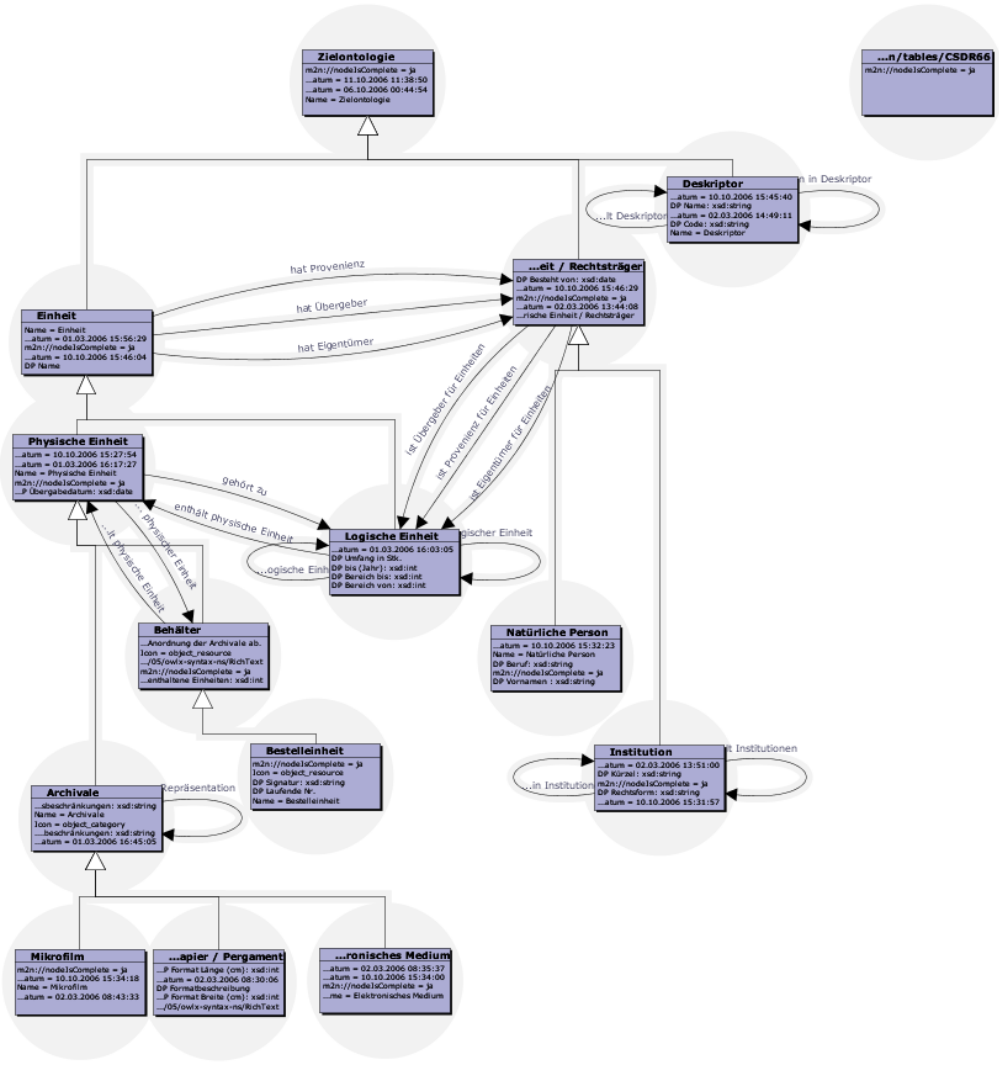


Figure 3.6: OntologyVisualization in the m2n Intelligence Management Framework 3.5

Chapter 4

Information Visualization

Information visualization is an interdisciplinary field which deals with the visual depiction of abstract data in order to facilitate exploration and analysis tasks (Spence [2000]; Hearst [2004]). Techniques developed in this field should help users to manage the every day flood of information, whereby interactive exploration is a key issue (Andrews [2002]).

Abstract data describe information where the visual display is not inherent, but is carefully chosen depending on the problem domain or task of the user. This is the major difference to scientific visualization where the visual representation is very often given. A computer tomograph data set, for example, as used in scientific volume visualization, contains items which can be described by their position in space and their density. Based on this information it is possible to reveal the original appearance of the scanned object in order to visualize bones or skin. The discussion of abstract data can be facilitated using the following *abstract datatypes* (Shneiderman [1996]; Andrews [2002]):

Linear/Temporal data contain information given in a sequential order. Shneiderman refers to this data as one-dimensional data and mentions lines of text as an example. Each of the lines may be given additional attributes like for instance font size, font color and font weight. Another example for linear representation is an ordered list. In temporal data the items are dependent on time like the throughput of a network device.

Spatial data have two or three dimensions. Each item might be represented as a two-dimensional shape or three-dimensional object. Tasks related to spatial visualizations deal with recognizing relationships and containment of one item in another.

Multi-dimensional data sets describing each item as a set of two or more properties. Such data is also referred to as *multi-variate* data.

Network data contains discriminable entities related to each other. In other words, the inherent structure is a directed or undirected graph (Herman et al. [2000]).

Hierarchical Data Sets can be thought of as a special kind of network data. In this case, the relationships between the entities describe containment or ordering.

The exact classification of data into these categories is often ambiguous. For example, semantic graphs can obviously be regarded as network data. Resources form the entities of the network and properties represent the edges. However, when assuming a frame and slot view, resources can also be regarded as multi-dimensional data, where each slot is a new dimension of a resource. In cases where the graph describes a taxonomy or inheritance hierarchy, the data can also be regarded as hierarchical data.

However, the datatype taxonomy is still helpful to understand and classify existing visualization methods. Thus, these categories will be referred to where appropriate. The remainder of this chapter

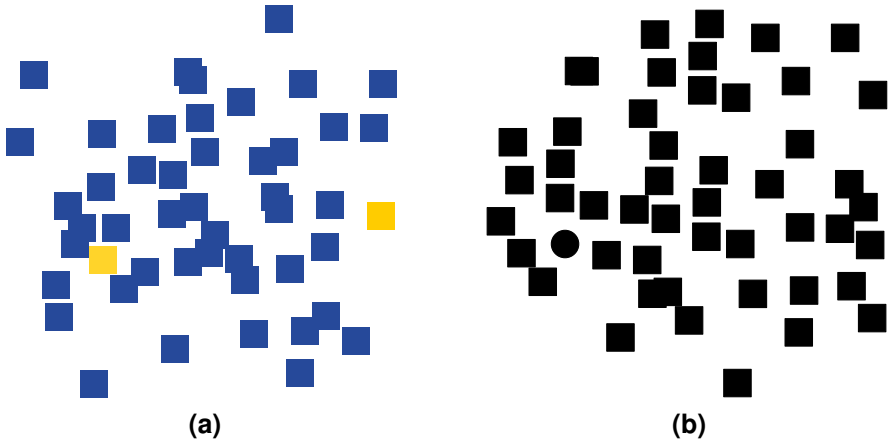


Figure 4.1: An example of preattentive color (a) and shape (b) recognition. The two features, color (yellow) and shape (circle), should be visible in less than 250ms.

gives an overview of techniques and terms having an influence on this thesis. The first part explains general techniques and terms used in information visualization. Afterwards, examples are given of network and hierarchy visualizations, since they are the most useful view of semantic graphs.

4.1 Preattentive Features

In order to simplify exploration and analysis tasks, a visualization may exploit *preattentive* features of the visual system of humans. Such features are immediately visible to the user or can be found with very high precision in complex displays without further cognitive effort (Healey et al. [1995]). Preattentive features do not require scanning of the visualization and can therefore be recognized in less than 200–250ms. Figure 4.1 gives an example for preattentive color and shape recognition. Both images contain squares that serve as a distraction pattern. Figure 4.1(a) uses color coding to drive the attention of the viewer. Figure 4.1(b) contains a circle that should be recognized immediately. The same effect could be achieved using other preattentive features like direction, texture, motion, and depth of field.

Some preattentive features seem to have a higher priority than others. For example color seems to be noticed with higher precision than shape. However, even if these features can be valuable to drive the attention of the viewer, the designer of a visualization must be aware that their combination is usually *not* preattentive. Extensive research about the perception in visualization has been done by Christopher G. Healey and can be found in Healey [2007]. The website also provides a program for testing preattentive perception of color, shape and their conjunction. Another reference about this topic is Ware [2004].

4.2 Linking & Brushing

Linking & brushing is an interaction technique which is used to couple multiple views of the same data set. When a change is applied to one of the views, it has an influence to the others (Keim [2002]). The underlying assumption is that it is possible to overcome the flaws of one representation by the strengths of another. Figure 4.2 shows the Coordinated Graph Visualization application presented in Abello et al. [2007]. Therein, the central view is coupled with the other views such that the selection of the node "einstein, e, mc" is synchronized with the tree (left) and the overall hierarchy tree (top).

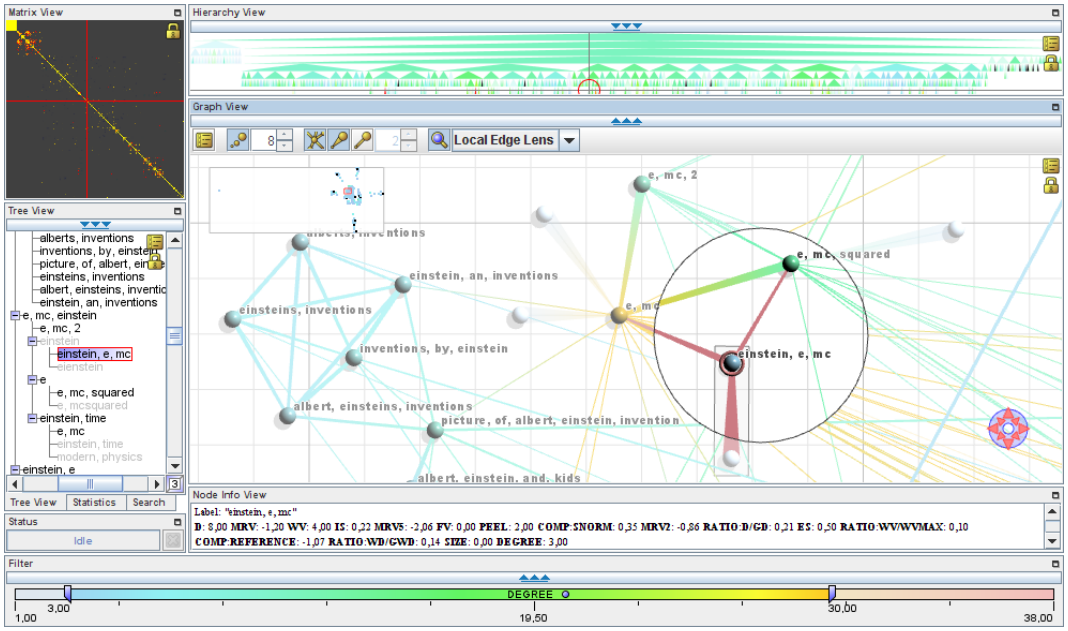


Figure 4.2: The Coordinated Graph Visualization (Abello et al. [2007]) application is an example for intense use of linking & brushing. The graph in the center is linked to the hierarchy tree on top and to the left. Used under the terms of the Austrian Copyright Law (UrhG) §46.

4.3 Interaction Techniques

A common paradigm in information visualization and user interface design is *direct manipulation*. This term was introduced by Ben Shneiderman in 1983 and describes a paradigm in the design of user interfaces (Shneiderman [1983]): a user interface should support the user to interact directly and reversibly with the elements on the screen while the application provides appropriate feedback.

Shneiderman also offers another paradigm which should be considered when designing information visualizations. The visual information seeking mantra states *"Overview first, zoom and filter, details on demand"* (Shneiderman [1996]), which means that any visualization should support the task of a user by providing an overview of the data. Next, the user should be able to apply filters on the data. Lastly, it should be possible to reveal detailed information about any item.

4.4 Focus & Context Techniques

A common problem in information visualization is that a user might loose the context of the presented information while investigating the details (Card et al. [1999]). This problem is tackled by focus & context techniques. Three different categories of these techniques can be distinguished: *distortion-oriented techniques*, *view and layer techniques* and *in-place techniques* (Hauser [2005]).

4.4.1 Distortion-Oriented Focus & Context Techniques

The overall idea of distortion-oriented focus & context techniques is to apply a distorting magnification effect to the area of interest, such that it appears in more detail. The behavior is best described using the stretching rubber-sheet, or magnifying glass metaphor (Sarkar and Brown [1992], Leung and Apperley [1994]). All items are drawn on a flexible rubber sheet which is stretched at the points of interest. Thus, items near these positions appear larger to the viewer. Usually two functions are applied to achieve this

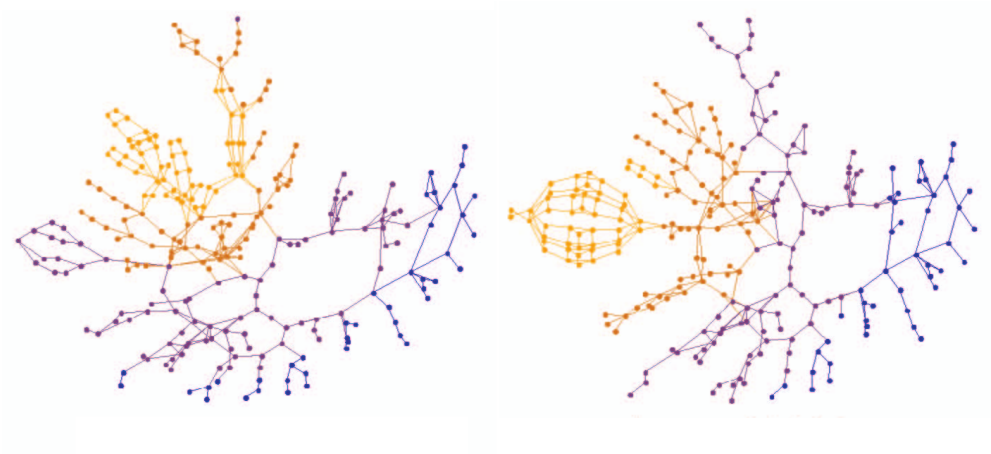


Figure 4.3: Topological fisheye views showing the focus on top and in the left part. Image extracted from Gansner et al. [2005]. Used under the terms of the Austrian Copyright Law (UrhG) §46.

effect. A distortion function maps the undistorted image to a distorted one and is responsible for the shape of the focus. A magnification function is used to control the amount of magnification and might be controllable by the user.

Popular examples for distortion-based techniques are fisheye views (Furnas [1986, 2006]), the Perspective Wall (Mackinlay et al. [1991]), the Table Lens (Rao and Card [1994]) or Hyperbolic Trees (Lamping et al. [1995]). Moreover, several deviations of the fisheye view approach exist. Some of them only work with network data like topological fisheye views (Gansner et al. [2005]) and semantic fisheye views (Janecek and Pu [2002]; Janecek [2004]).

Topological fisheye views allow the investigation of huge graphs. The underlying algorithm calculates a hierarchy of coarsened versions of the original graphs. To achieve this, the nodes are clustered with respect to their topological distance in the original graph. The number of edges between the nodes is taken into account in order to determine the level of detail in the focal region. When a focus is set, a horizontal slice is selected from the hierarchy which shows a hybrid version of the graph containing nodes and clusters. Topological fisheye views can handle huge graphs containing more than $100,000$ nodes. However, in the context of this thesis, it is questionable if the topological distance between nodes is a reasonable metric.

Semantic fisheye views are another variation which measure the distance of nodes with respect to the underlying data model but not its final representation (Janecek and Pu [2002]; Janecek [2004]). However, this implies some reasonable distance metric for the entities in the data model. In the case of ontologies this issue is still a field of research.

Similar to semantic fisheye views and toplogic fisheye views, hyperbolic trees are not applicable to general data but require hierarchical datasets, because the layout algorithm relies on a tree structure (Lamping et al. [1995]). The layout is calculated in a two-step process starting at the root node of the tree. In the first step, the tree layout is calculated in the hyperbolic plane. Each node is assigned a wedge whose tip is placed at the node itself. All children of that node are placed at equal distance from the node on the arc of the wedge. This process is repeated for each child such that each branch of the root is assigned an equally sized wedge. The result of this process is mapped to the unit disc in the euclidean plane. Figure 4.4 shows the result for a non-uniform tree with overlaid labels.

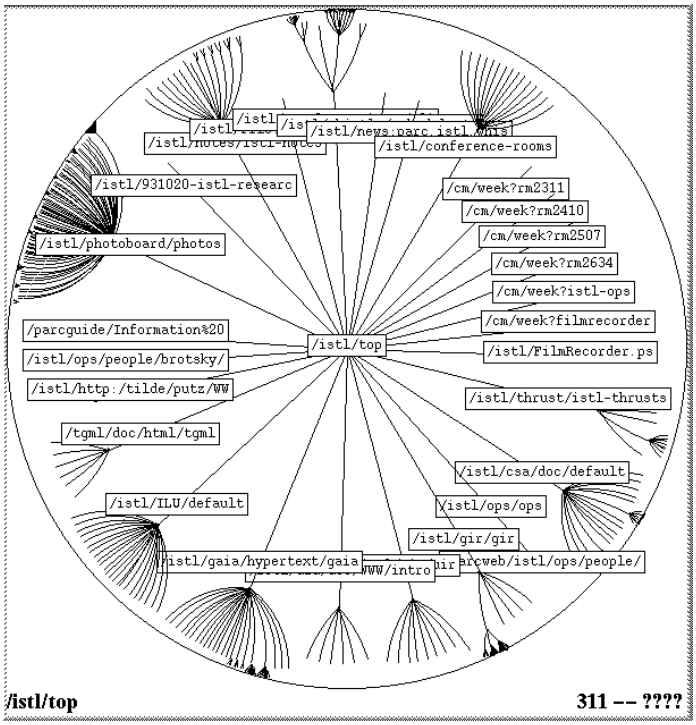


Figure 4.4: Hyperbolic Tree. Image extracted from Lamping et al. [1995]. Used under the terms of the Austrian Copyright Law (UrhG) §46.

4.4.2 View and Layer-Based Focus & Context Techniques

View and layer-based techniques provide the detailed information or the context information in a second view or by overlaying the data in question. For example, image manipulation tools or cartography applications usually provide an extra view which provides a *bird's eye perspective* of the image or the map. Depending on the application, this extra window provides different capabilities, such as positioning the viewport or zooming.

Excentric labels, introduced in Fekete and Plaisant [1999], are another example of a layer-based technique. This method overlays all labels within the local neighborhood of the focal point which is determined by the current mouse position. The labels are placed distant to the actual data item but are connected to them via lines, as shown in Figure 4.5. Fekete and Plaisant [1999] describe different versions of the label placement with respect to the number of line crossings and label closeness to the linked item. For instance, the labels shown in Figure 4.5 preserve the vertical order of the linked items. Drawbacks of this approach result from missing or very long labels. The maximum number of labels shown should also be restricted to 20 at most since the display is easily cluttered otherwise.

Another example of a layer-based technique is the see-through interface introduced in Bier et al. [1993]. This interface is made of a set of widgets placed on a semi-transparent sheet over the data. The user needs two hands to position and use the widgets. One hand is used to place the widget and the other one is used to apply the actual function. The example presented by Bier et al. [1993] demonstrates this approach in an image manipulation program which provides widgets for applying fonts, colors, and local lenses. It is also pointed out that different tools can be overlaid and combined.

4.4.3 In-Place Focus & Context Techniques

In-place focus & context techniques use visual features to drive the attention of the user. An example is the *semantic depth of field* approach presented in Kosara et al. [2002]. The idea of this method is to apply

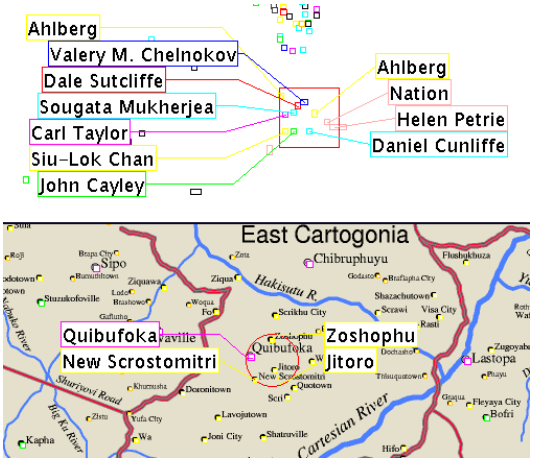


Figure 4.5: Screenshot of excentric labels applied to a scatterplot (top) and overlaid on a map (bottom). Image extracted from Fekete and Plaisant [1999] under the terms of the ACM copyright. ©by Association for Computing Machinery, Inc.

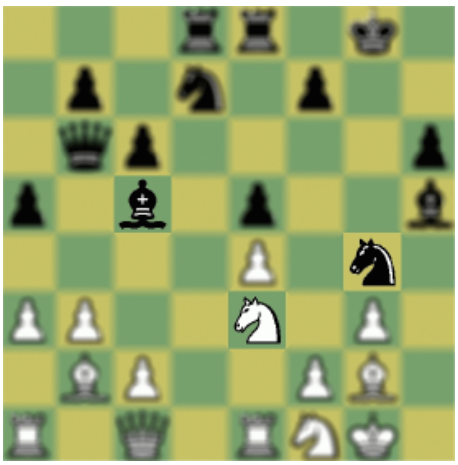


Figure 4.6: Semantic depth of field is an example of a preattentive in-place focus & context technique. Attention is driven automatically to the in-focus areas. Image extracted from Kosara et al. [2002]. Used under the terms of the Austrian Copyright Law (UrhG) §46.

a blur effect to areas distant from the focal point. Figure 4.6 shows this method applied as a chessboard tutor system. The white horse and the black horse threaten each other which is immediately visible due to preattentive perception of the depth of field effect.

Another example of an in-place technique is GeoSpace (Lokuge and Ishizaki [1995]). Figure 4.7 shows a map around Cambridge. The red dots indicate crime, red squares represent hospitals, and the bright lines show streets. Figure 4.7(a) highlights Cambridge and shows context relevant information. In this case the question about streets leading to hospitals can easily be answered. The highlight is based on color, font size, and opacity. In contrast, this information hardly is recognized in Figure 4.7(b).

4.5 Network Visualization

The visualization of networks in information visualization is related to traditional graph drawing. Traditional graph drawing is mostly concerned with the presentation of a graph as a node-link diagram. The focus is put on planarity tests, layout algorithms, and the complexity and predictability of layout algorithms (Herman et al. [2000]). Information visualization emphasizes the interaction with a visu-

(a) (b)

Figure 4.7: GeoSpace is an example of an in-place focus & context technique. Red dots indicate crime, red squares indicate hospitals, and white lines show streets. (a) shows a filtered view of (b) and focuses on hospitals reachable from Cambridge. Image extracted from Lokuge and Ishizaki [1995] under the terms of the ACM copyright. ©by Association for Computing Machinery, Inc.

alization and tries to find many different representations of network data. This is due to the fact that most algorithms fail to produce good layouts and require a significant amount of time for a large graph. An example of the integration of information visualization techniques in common graph drawings are topological fisheye views, mentioned in Section 4.4.1.

Semantic graphs can be seen as networks of multivariate data. A visualization technique which can be used to display such graphs is called *PivotGraph* (Wattenberg [2006]). The PivotGraph groups and filters the underlying data with respect to selected properties. This way, comparisons of the number of links should be possible. Figure 4.8 shows an example comparing the number of contacts between men and women at five different locations. Obviously, the most relationships exist at Location B. The PivotGraph approach seems to be promising, since instances in a RDF graph could be categorized easily with respect to their classes.

4.6 Hierarchy Visualization

With respect to semantic graphs, hierarchies are important when it comes to the visualization of inheritance relationships in an ontology. Node-link diagrams usually make poor utilization of space with respect to the displayed information. Treemaps try to solve this problem by representing the hierarchy through nested rectangles which are subdivided recursively (Johnson and Shneiderman [1991]). The subdivision alternates in horizontal and vertical directions. The space assigned to a rectangle corresponds to the number of children making it easy to see quantitative relationships. The underlying idea is similar to Venn Diagrams used to represent containment in set theory. Treemaps are well suited for tasks requiring the detection of containments. Figure 4.9 shows the basic principle of the treemap and its related tree.

The *SpaceTree* shown in Figure 4.10 tackles the same problem, but tries to use space more efficiently by collapsing unused branches (Plaisant et al. [2002]). Only the currently selected node is expanded. If a node has children but is collapsed, a glyph appears in order to indicate subtrees. Depending on the size of the subtree, a triangle or a miniaturized version is shown. A basic interaction mechanism is the automatic folding and unfolding of nodes. If a new node is selected, the first step is to collapse the possibly expanded subtree. Afterwards, the viewport is animated to center the selected node, and finally the new branch is expanded. The animations are intended to preserve the context while the tree is being explored.

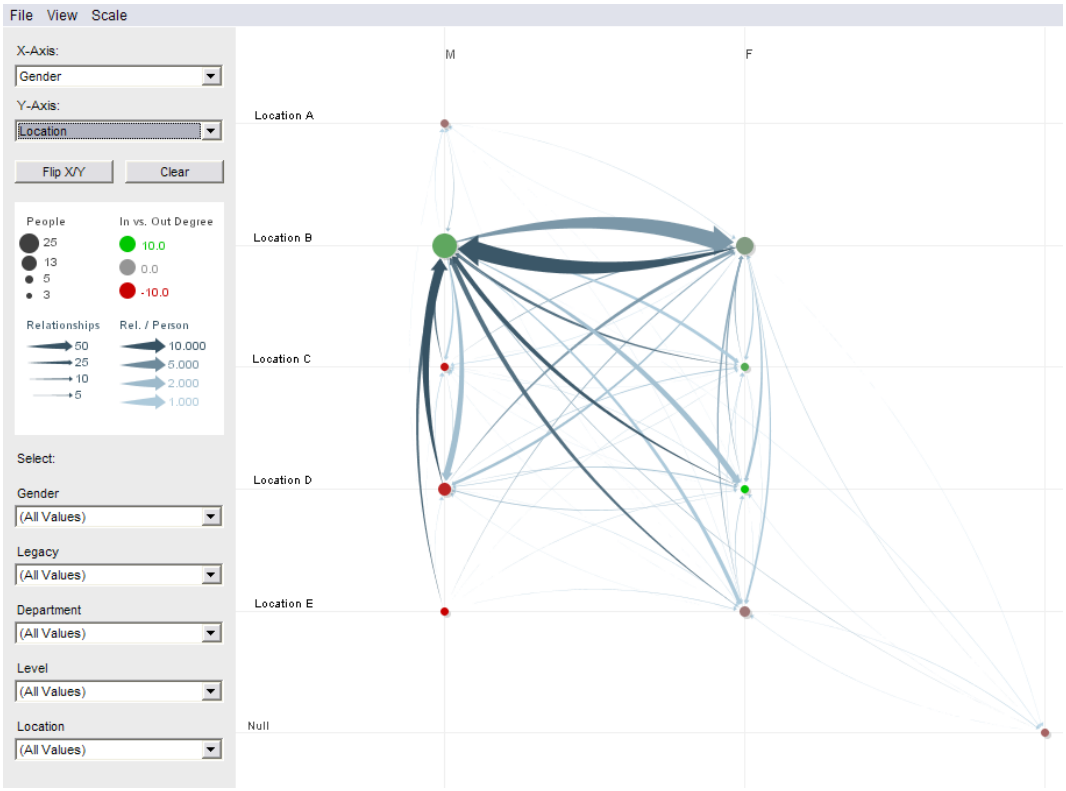


Figure 4.8: Example of a PivotGraph for the visualization of multivariate network data. The first column shows the number of men at five locations. The second column shows the number of females at the respective locations. The strengths of the arcs indicate the number of contacts between the locations. Image extracted from Wattenberg [2006] under the terms of the ACM copyright. ©by Association for Computing Machinery, Inc.

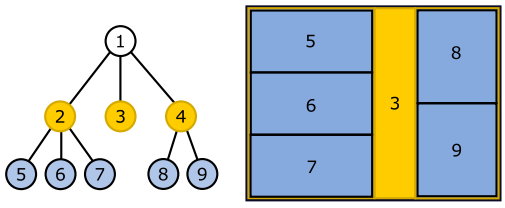


Figure 4.9: Example of a tree and the corresponding treemap visualization. The root node of the tree is mapped to the framing rectangle. It is subdivided recursively for each level in the hierarchy, first vertically, then horizontally. The colors are not mandatory but make the concept clear.

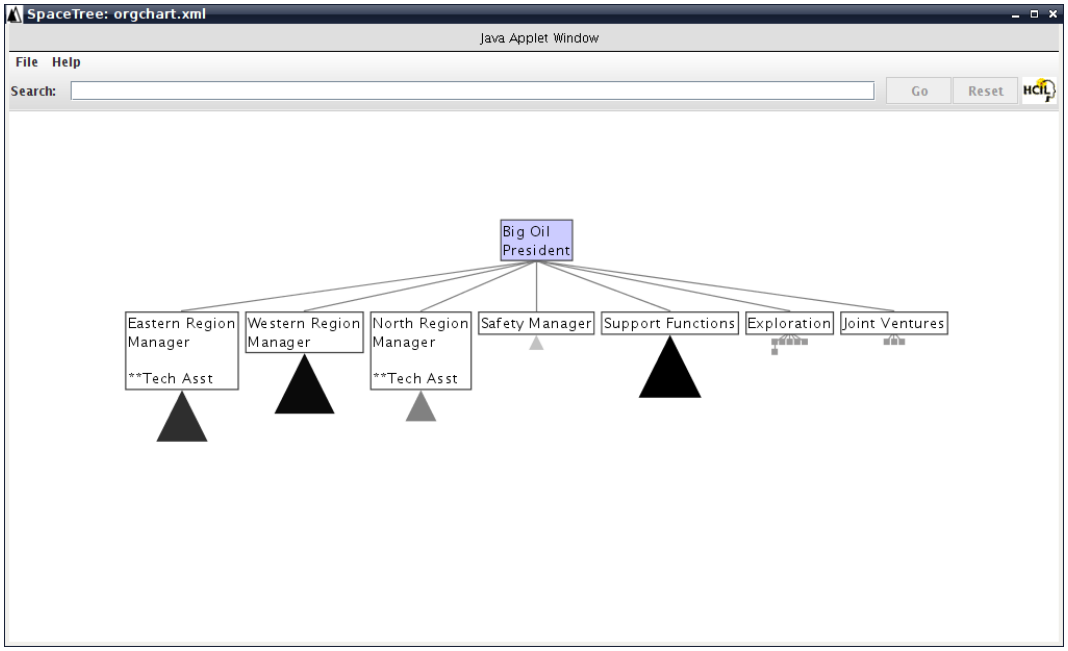


Figure 4.10: Example of a SpaceTree. The SpaceTree tries to utilize space more efficiently by automatically collapsing unused branches. Instead, glyphs are shown to indicate existing data. Image extracted from Plaisant et al. [2002]. Used under the terms of the Austrian Copyright Law (UrhG) §46.

4.7 Zoomable User Interfaces

Zoomable User Interfaces (ZUI) were first described by Perlin and Fox [1993] in order to explore large information spaces. In such interfaces, the information objects are placed in an infinite two-dimensional plane. For exploration, the user navigates and zooms to the objects of interest. The transitions between the zoom levels or navigation points are animated. ZUIs often provide two metaphors: *semantic zooming* and *portals*.

Semantic zooming provides a way to hide complexity based on the scale and size of an information entity (Bederson and Hollan [1994]). The higher the zoom level, the more detail is shown. For example, suppose a document in the information plane. If low interest is assigned to this document it could appear as rectangle. If the user zooms to the document or enlarges its bounding rectangle the title and successive details of its contents become visible. In this sense, semantic zoom provides a *levels of detail* mechanism. The problem related to these detail levels is to find the appropriate zoom levels which cause the change of the presentation. However, it is questionable whether such functions exist in general, since they depend on the content of the visualization.

The second commonly used metaphor in ZUIs are *portals* (Bederson et al. [2004]). A portal provides a view on another part of the information plane. This concept is borrowed from 3D computer graphics where a scene is represented as a graph. A portal is a camera which looks at a specific part of this graph.

4.8 Summary

This chapter gave an introduction to information visualization and introduced common visualization techniques like focus & context and linking & brushing. Distortion-oriented techniques seem to be promising at first sight in the reduction of screen clutter. However, as shown in Lau et al. [2004], these techniques put the cognitive effort on the user to re-assemble the focus and the context. Even though their

results show that this effort is negligible, it must be noticed that only static images had been used in their experiments. No complex interaction was required which could have significant impact on performing tasks. For example, as Lamping et al. [1995] point out, the hyperbolic tree tends to rotate the nodes around the center when they are distant to the focus. This is somewhat unnatural and could influence the user's orientation, when comparing research in other fields like Kosslyn [1998].

Other techniques, such as linking & brushing seem to support the visualization in a very general way. The potential of these techniques is their applicability in multiple domains. To find out which techniques are actually used in real world applications, the next chapter will discuss recent semantic graph manipulation and visualization tools.

Chapter 5

Semantic Graph Visualization and Manipulation Tools

This chapter reviews recent visualization and manipulation tools for semantic graphs. The tools are described with respect to their presentation and their support for orientation, navigation and manipulation of the graphs. As far as possible, the programs have been tested. In all other cases, the description relies on the respective publications and program documentations.

Many tools are available which are only capable of visualizing RDF graphs. An example is RDF Gravity developed at Salzburg Research (Goyal and Westenthaler [2004]). The program is capable of visualizing and browsing RDF data using node-link diagrams using a dynamical layout. Another example is ISWIVE which uses topic maps for the exploration of semantic graphs (Chen et al. [2005]). However, this review only includes tools which also allow the interactive editing of RDF graphs. A more comprehensive study, which also considers aspects of software usability, can be found in Katifori et al. [2007]. However, the field was rapidly changing during the time of writing, so that this review might be out of date already.

5.1 IsaViz: A Visual Authoring Tool for RDF

IsaViz is an open source Java tool for visualizing and manipulating RDF data developed by Emanuel Pietriga. The current version of the program can be found at `www.w3.org/2001/11/IsaViz/`. IsaViz shows a flat version of the triples stored in a RDF graph as a node-link diagram. The presentation is similar to the graphs shown in RDF-Tutorials like Manola and Miller [2004]. Properties appear as blue arcs with a label attached, resources appear as green ellipses including their URI and literals are shown as yellow rectangles. This default appearance can be changed using graph style sheets. These styles offer great flexibility for modifying the presentation. For example, it is also possible to define new glyphs for nodes. The appearance of nodes can become quite complex and encode more information than simply the URI.

Figure 5.1 shows the four main windows of the program in the default configuration. Besides the actual visualization, IsaViz provides an *editor panel*, an *attribute panel* and a *definitions palette*. The editor provides several tools to select, create, and manipulate the graph. The definitions palette is used for creating and importing properties. The attribute palette shows the literal values of the selected nodes.

5.1.1 Orientation

IsaViz is a ZUI which uses the Zoomable Visual Transformation Machine (Pietriga [2005]). Double-clicking a node animates the viewport such that the node is centered in the main window. In order to

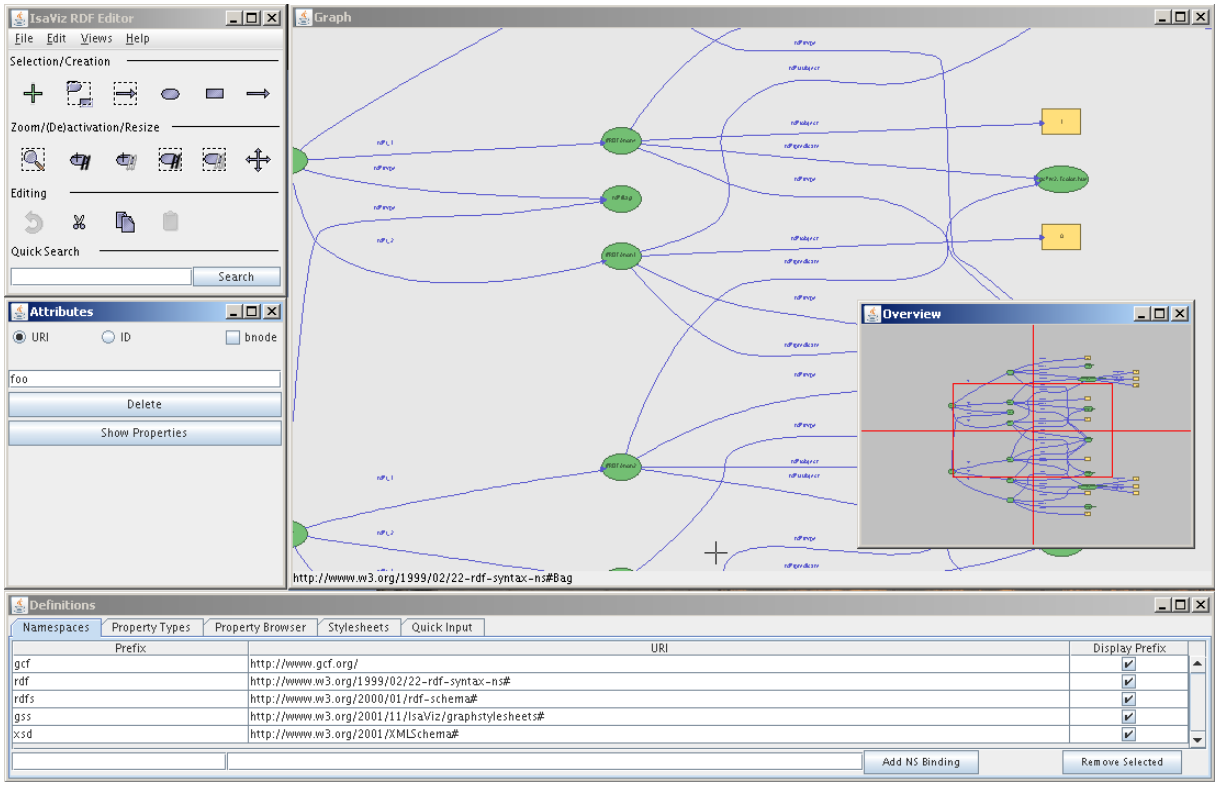


Figure 5.1: Main window of IsaViz in its default configuration including the editor panel, the attribute panel, the definitions palette, and the overview window called the "radar view".

move the viewport the mouse must be dragged in the respective direction. As long as the mouse button is held down the viewport moves. Meanwhile, the speed of movement is visualized as an arrow. The same applies for zooming except that the shift key must be pressed while dragging the mouse up or down.

For orientation purposes an overview window called the "radar view" is provided. Pietriga [2005] also mentions that a fisheye view is available. However, it could not be examined how it can be activated. Another feature is the possibility to search for resources using the quick search field in the editor palette. After a search the viewport focusses on the corresponding resources. If more than one result is found, it is required to search a second time. A more advanced search is also possible where the user can restrict the search to properties, resources, or only literals.

5.1.2 Creation and Manipulation

In order to create new resources the user selects the resource tool from the editor palette and clicks into the visualization. This brings up a dialog where the URI of the created resource can be entered. For further manipulation the attribute palette must be used. To create properties the user is given another tool in the editor palette. The creation follows three steps: Firstly the type of the new property must be selected from the definitions palette. Secondly, the subject node must be selected. Lastly, the object must be selected. A hint is shown at the bottom of the main window to clarify which part of the statement is required. If the creation is successful, the edge appears.

5.2 Jambalaya 2.6

Jambalaya 2.6 is a Plug-In for Protégé developed by the Chisel Group (Noy et al. [2000]; Storey et al. [2001, 2002]). The current version of the Plug-In can be found at `http://www.thechiselgroup.`

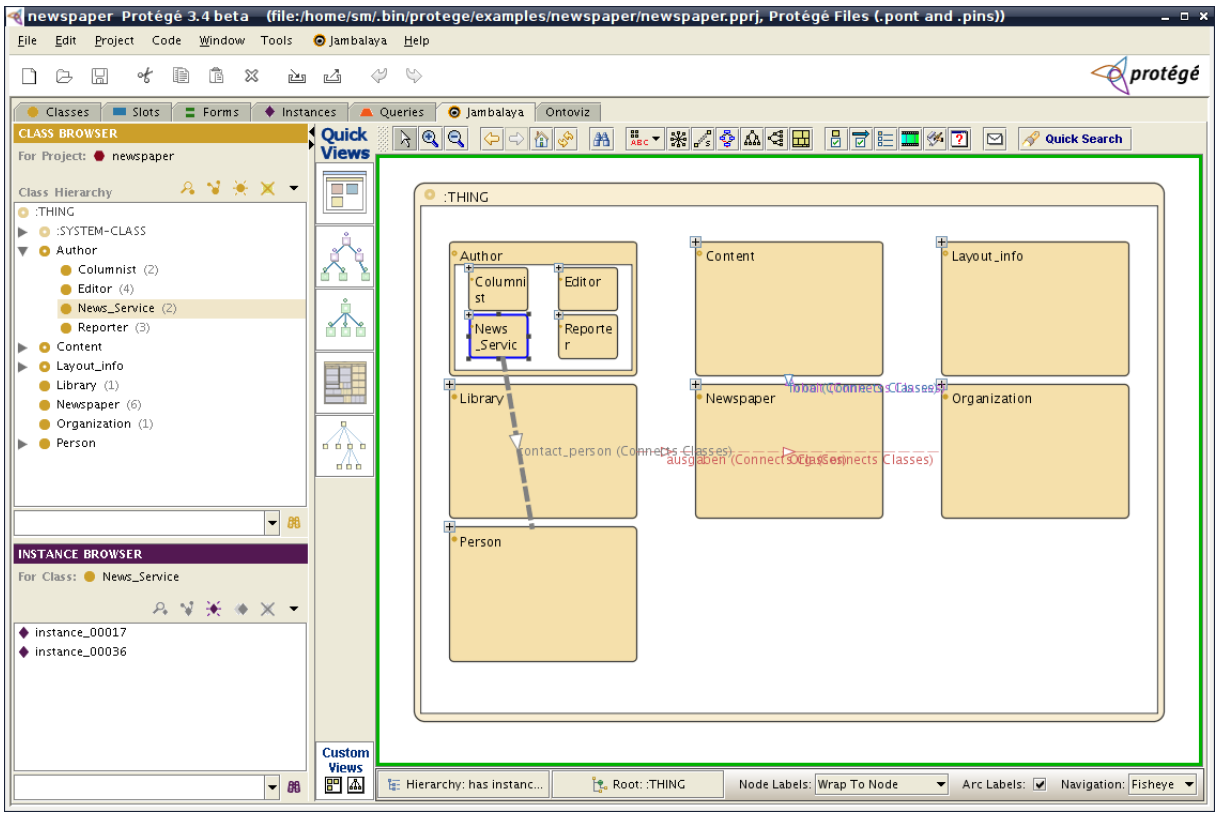


Figure 5.2: Screen capture of the Jambalaya Plug-In in Protégé 3.6. By default the visualization opens a treemap representation.

com/. Jambalaya supports the exploration and manipulation of ontologies by the means of a ZUI which relies on the Piccolo framework (Bederson et al. [2004]). Figure 5.2 shows the default view of an OWL ontology provided as an example with Protégé. At the top of the visualization window, Jambalaya provides a toolbar, the left region defines buttons to apply different modes of presentation, and a status bar lies along the bottom.

By default, the classes of the ontology are represented as a treemap using the subclass property as the hierarchy criterion. However, since OWL allows multiple inheritance, some classes may appear twice. In order to emphasize their identity, Jambalaya paints these classes with a blue border if any of them is selected. The treemap visualization also shows the object properties as rounded curves including their name.

The left panel is particulary interesting, since it provides support for the definition of different presentation setups. For instance, it is possible to view hierarchies as a treemap, as a tree, or as a flat node-link representation. Each new view appears as a button on the left panel and can be activated at any time.

5.2.1 Orientation

Jambalaya uses the ZUI metaphor in a more extensive way than IsaViz. For example, the default treemap visualization of an ontology supports multiple levels of detail (semantic zoom). By default, the treemap shows only the children of the root and hides the respective subclasses. Further investigation requires a double-click on the class or zooming in. In this case, the level of detail is increased and shows further information such as the grandchildren of the root node. An important characteristic is that the viewport always follows the actions of the user and tries to center the selected class.

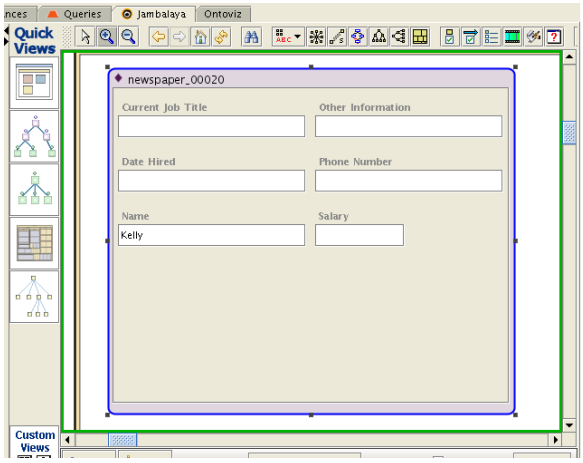


Figure 5.3: An example for In-Place editing of an instance in Jambalaya. When double clicking a node, Jambalaya opens the inline editor of Protégé. This editing is not possible on edges.

Further orientation support is provided by an overview window. Using this overview, it is also possible to change the magnification by dragging the edges of the viewport indicator. In addition to this, Jambalaya provides several search features. Using the quick search button in the toolbar it is possible to perform an incremental search. The names of possible resources in the visualization are proposed as typing proceeds. The viewport zooms out in order to fit in all possible results.

5.2.2 Creation and Manipulation

Editing a class or an instance requires setting the maximum zoom level, so that the visualization of the respective resource fills the available viewport. In this state Jambalaya will automatically show the default editor of Protégé as shown in Figure 5.3. This approach does not force the user to switch to another view but does have other drawbacks: The inline editing of object properties is not possible this way since edges do not have visible bounds to place the editor. Furthermore, it is not possible to edit an element which is not at the bottom level of the hierarchy. For example, the class Author in Figure 5.2 can not be edited using full zoom, since this expands the nodes to show the underlying classes and hence never provides an editor. Finally, the animation takes an unnecessarily long time to complete.

5.3 OntoTrack

OntoTrack is an ontology authoring tool which visualizes ontologies and their instances using an adapted space tree. The editor is designed to work together with a reasoner which is not available for free. Therefore the following description is based on Liebig [2003]; Liebig and Noppens [2004] and the online manual (OntoTrackManual [2009]). Figure 5.4 shows the main window of the program. The goal of this tool is to combine the graph visualization approach with direct manipulation methods using editing modes.

The main window of the program consist of three parts: A *menu bar*, a *search bar*, and the *visualization area*. The latter is used to browse and manipulate the graph. As shown in Figure 5.4, the program can handle multiple ontologies at once. Each visualization is put into an extra tab at the bottom. OntoTrack assumes that the most important relationship in an OWL ontology is the subclass relationship. More precisely, only direct superclass relationships are taken into account. All other properties are not shown in order to preserve the tree structure. Hence, the nodes in the space tree either show classes or

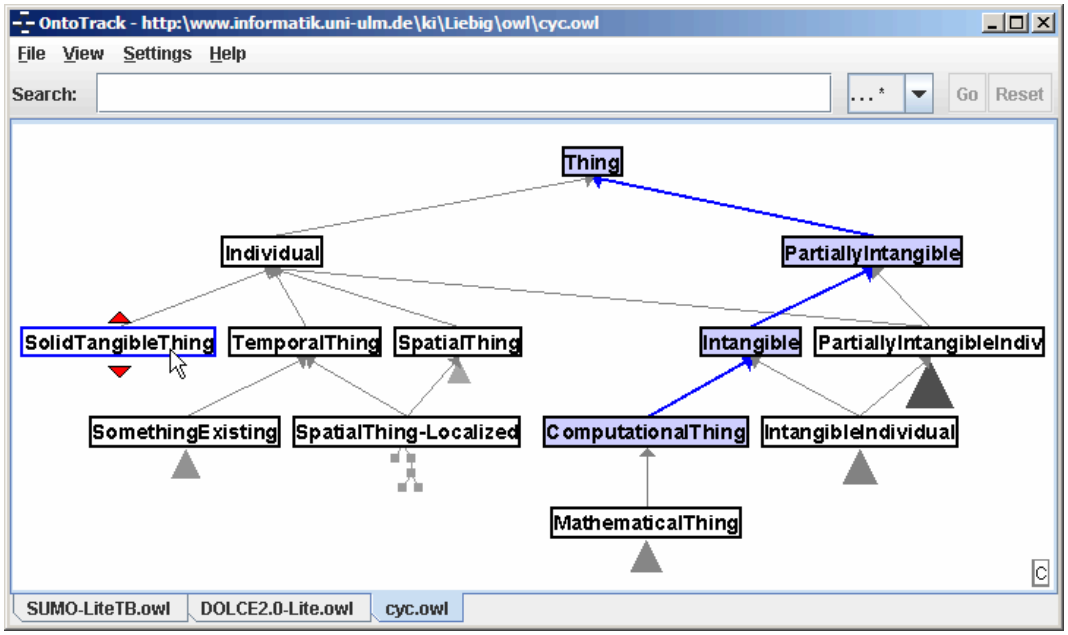


Figure 5.4: The main view of OntoTrack. Image extracted from Liebig and Noppens [2004]. Used under the terms of the Austrian Copyright Law (UrhG) §46.

reified properties of an ontology. Like the treemap in Protégé, the space tree needs auxiliary structures to preserve the tree structure for the case of multiple inheritance edges.

5.3.1 Orientation

OntoTrack provides a search field for searching for instances or classes in the displayed graph. The navigation follows the space tree description given in the previous chapter (Section 4.6). The viewport is animated in order to center the current selection, and the branches of the tree are automatically contracted and expanded. Moreover, the path from the last expanded parent to the root is highlighted. OntoTrack always tries to fit the whole hierarchy into the viewport. Only if that fails, an overview window does become visible.

5.3.2 Creation and Manipulation

OntoTrack provides nearly complete support of the OWL-Lite vocabulary. Editing is started using different handles shown beside the nodes of the tree. The availability of such handles depends on the current edit mode. For example, in order to create a new subclass for SolidTangibleThing in Figure 5.4, the red triangle at the bottom needs to be dragged with the mouse. To create new slots for a class, the editor needs to be switched to the detailed presentation mode. In this case, nodes are shown with their slots and allow inline editing.

5.4 Ontorama

Ontorama is a Java program designed for browsing and manipulating RDF-based ontologies as node-link diagrams in a hyperbolic layout (Eklund et al. [2002, 2006]). Properties are shown as simple colored arcs with overlaid icons. Nodes appear as blue circles overlaid with their URI. As outlined in Eklund et al. [2002], the hyperbolic layout causes some issues, since ontologies in general are not trees but general graphs. In order to recreate the tree structure of a graph, nodes with multiple incoming edges are cloned

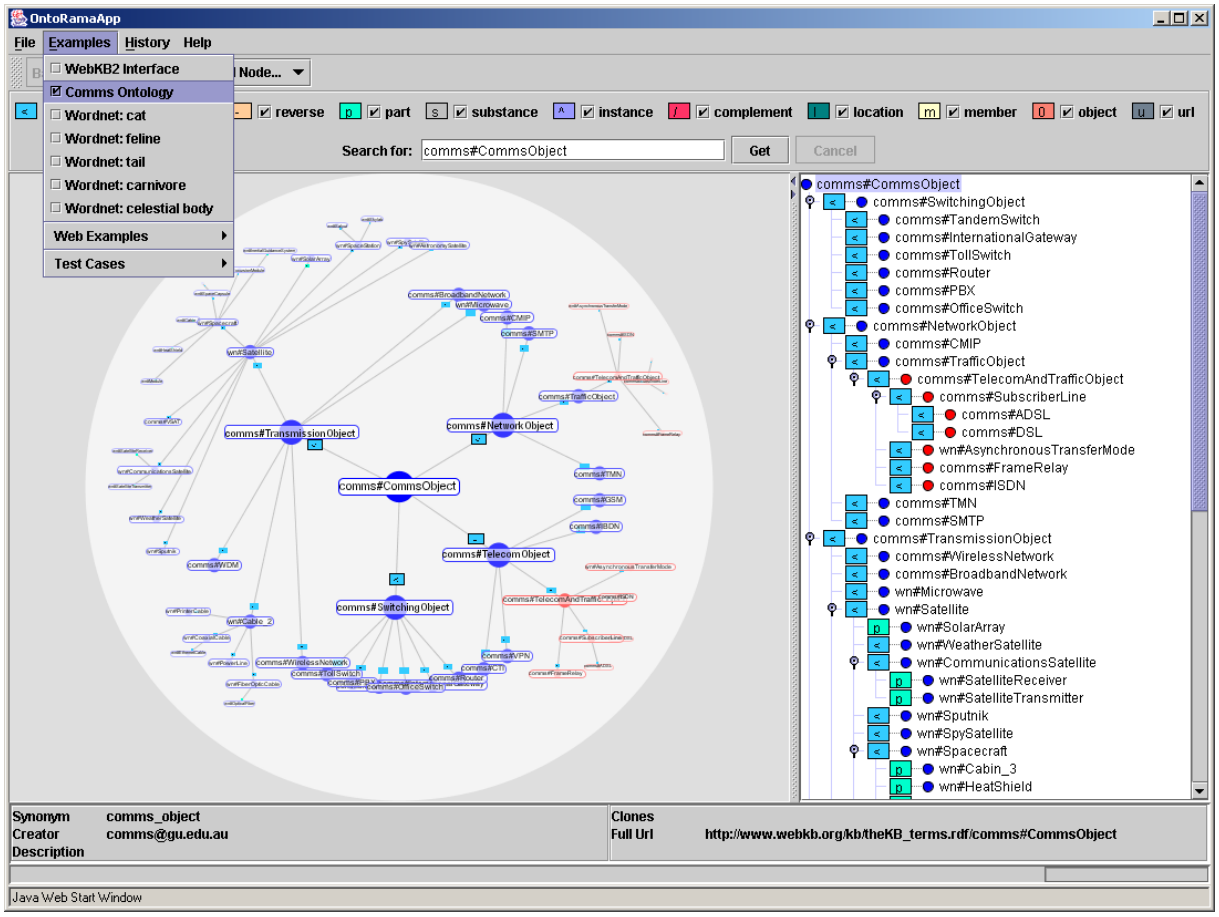


Figure 5.5: Ontorama supports the collaborative manipualtion of OWL ontologies using a hyperbolic tree (Eklund et al. [2002]). In order to preserve the tree structure, cloning of graph structures is necessary. Cloned structures appear in red.

including their subgraphs. Such cloned structures are drawn in red and are connected with an additional edge. These auxiliary structures are more complex than those introduced in OntoTrack or Protégé as Figure 5.5 shows.

5.4.1 Orientation

Ontorama provides some features which are visible in Figure 5.5 and are worth mentioning. The left panel of the main window contains a tree view of the graph which is synchronized to the visualization. When selecting a node in the hyperbolic view, the respective node in the tree is selected as well. Another feature is the history of selected nodes. The user can quickly jump to past selections by choosing them from the menu. Like the other tools, Ontorama provides a quick search feature accessible from the toolbar.

5.4.2 Creation and Manipulation

Ontorama is designed as a peer to peer ontology editing tool, so that multiple users can edit the same ontology at the same time. A new resource can be created using the context menu. The entries of the menu depend on the selection and the visible nodes. However, from the description given in Eklund et al. [2006] it is not obvious whether properties can be created directly in the visualization. It is also unclear how creation influences the auxiliary structures described above.

5.5 Summary

This chapter reviewed tools used for the visualization and manipulation of RDF-based semantic graphs. Except IsaViz, all tools specialize on the modeling and visualization of ontologies. IsaViz allows the manipulation of RDF graphs in general applying several techniques from information visualization. OntoRama seeks to visualize RDF-based vocabularies using a hyperbolic tree. Jambalaya applies a treemap. OntoTrack uses an adapted space tree. While the use of a tree structure seems to be natural at first sight, auxiliary elements need to be added in order to support multiple inheritance. Moreover, the treemap makes it hard to read object properties as they often overlap. OntoTrack does not show object properties at all except subclass relations. Instead, another view of the property hierarchy is required. Compared to the UML like notation in the m2n IMF GraphVisualization, all representation forms seem to have more drawbacks than advantages.

Exploration and navigation tasks are mostly supported through search or an extra view which provides a bird's eye perspective. OntoTrack and Jambalaya also make extensive use of animations. This is beneficial for exploration of unknown data since it is easy to maintain context. However, animations are distracting when it comes to modeling and the user is forced to wait for the animation to complete. The full zoom metaphor of Jambalaya also forbids a switching to other resources, since only a single resource is visible at the time of editing. Thus, the m2n IMF GraphVisualization should support animations, but allow the user to interrupt or to disable them at any time.

The inline editing provided by Jambalaya is interesting, but as mentioned above, it is not applicable for object properties and requires switching to another approach. Therefore, the m2n IMF GraphVisualization should support editing as a mixture of the approaches used by OntoTrack and IsaViz.

Chapter 6

Design Goals

The goal of this thesis is the improvement of the GraphVisualization Plug-In (GVP) in order to simplify the exploration and modeling processes. Both processes have different target groups. While exploration is mostly focused on end users of an adapted client, modeling is more focused on developers. However, even developers must be grouped into those familiar with the technology and those with no prior experience. Modeling spans a large domain of applications. It includes the adaptation of the target ontology, the adaptation of rules, and the manipulation of arbitrary resources in the application graph.

This chapter gives an overview of the goals which influenced the design of the new GraphVisualization Plug-In. The next section explains how requirements were formed and what general requirements exist. Section 6.2 describes two hypothetical users of the framework who should be able to work with the component after the redesign. The scenarios which need to be supported are described in Section 6.2.1 and Section 6.2.2. Lastly, the GVP's conceptual model is presented, with which a user will work when performing tasks in the visualization.

6.1 Software Requirements

As mentioned in [Sutcliffe, 2002, p. 45], there is no "cook book" for collecting the requirements of a project. However, in most cases a project is started with a vague knowledge about the problem and its solution. In case of the GVP, management requested to "improve the visualization". It was expressed that the current implementation lacks orientation support, which might be improved by implementing a hyperbolic tree. However, as discussed in the last two chapters, the hyperbolic tree or distortion-oriented techniques are not regarded as the best solution due to the cognitive effort they put on the user and their dependency on tree structures. Thus, the first task was the collection of actual requirements through user research.

A common practice in requirements engineering is to collect information about the problem domain through interviews, questionnaires, observation, participation, and document analysis. The most insight into a problem domain can be achieved from participation. That is, performing the tasks a user is required to do. As part of the development team this was the most natural way to collect requirements. Further information is based on informal interviews with colleagues and participation at project start meetings and code reviews. A major problem at the beginning of the project was the lack of documentation of requirements and code. The following informal requests were collected:

- "It is hard to preserve orientation when navigating in larger graphs. For example, when adding an object property it is required to zoom and pan often between multiple positions. This takes too much time."

- "It should be possible to find resources quickly in the visualization. Maybe it would be helpful to have some sort of bookmark?"
- "I would like to see at a glance how rule blocks and variables depend on each other. It is not possible since nodes and edges often overlap."
- "Modeling an ontology is tedious and should be improved. It should be possible to edit any resource shown in the visualization directly instead of using the DetailView and the MultiTree."
- "We have a customer who needs to build formula trees. Would it be possible to build formulas using the visualization?"
- "We have a customer who wants to visualize and change parts of their organizational structures which keep changing over time. It should be possible to show and change these structures using the visualization!"
- "The performance should be improved. When visualizing a rule or a schema with only 20 nodes, the rendering becomes amazingly slow."

The essence of these informal requirements is that the GVP needs to be adaptable for different applications in terms of the *contents* to be shown, the *presentation* of the contents and the *available manipulation and exploration* tools. Moreover, this application is used by a broad spectrum of users, all of them having different knowledge, goals, and understanding of semantic technologies.

In order to compile these requests into a more structured form, typical users of the visualization were identified. A user model was defined identifying different groups of users as *developers*, *consultants*, and *end users*. Based on the description of developers and the interim development of end user applications the common use cases of the visualization were analyzed to find drawbacks of the current visualization. This step was performed multiple times leading to more generalized use cases in each iteration. The final findings were used to define a conceptual model in the way proposed by Johnson [2008].

6.2 User Model and Usage Scenarios

An often cited principle in user interface design is to know the users. That means users should be described in more detail than simply "novice" and "expert" ([Johnson, 2008, p. 11]). For example, it is possible for a senior programmer to fail to create a complex spreadsheet which is no problem for a secretary. This is because developers are faced with different tasks in their daily work. Therefore, a more detailed description of users must rely on further characteristics. For the redesign the following criteria were used to characterize target users:

Position: This property describes a user's job. Potential values include management, development, or a more specific description.

Computer Usage: This dimension describes the purpose of the daily use of the computer. For example, secretaries will mostly use office programs whereas developers will typically use a development environment.

Technology Background: The technology background specifies the familiarity with concepts of software development, object orientation, and semantic technologies like RDF and OWL. Further background knowledge might result from the use of modeling tools like Protégé. The lack of technology background may prohibit modeling, but should not prevent using the visualization for exploration purposes.

Framework Knowledge: This dimension describes the background knowledge of the user about the m2n Framework. If the framework is known, familiarity with a prior version of the GVP is of interest.

Visualization Use: This dimension reflects the purpose a user would utilize the visualization. For example, developers will most likely use the visualization for modeling the target ontology.

For the framework three main target user groups were identified: *developers* of m2n, *consultants* of external companies, and *end users*. Cooper [1999] proposes to find *personas* in the target groups to design the product for their needs. A persona is not a real person but an archetype of a user combining different characteristics and goals. The program should seek to support the persona in performing their tasks. For the redesign of the GraphVisualization Plug-In the following two archetypes were built: Tom Nelson and Sue Oliviera.

6.2.1 Tom Nelson

Position: Tom works as a developer at m2n. His responsibilities include the development of core components and new features, quality assurance, test, and documentation of implemented functions.

Computer Usage: Tom mainly uses development tools like Eclipse for writing Java code. For the adaptation of N3 files, Tom mostly resorts to simple text editors.

Technology Background: Tom is familiar with semantic technologies like RDF and OWL. He knows parts of the m2n schema, but not all.

Framework Knowledge: Tom has a deep understanding of the framework including implementation details. He is familiar with the concept of rules and the target ontology. However, he has not previously used the visualization for modeling.

Visualization Use: The main use case for Tom is the adaptation of the target ontology, the inspection and adaptation rules, and the creation of schema, such as sub-graphs which are hard to handle in N3 files.

6.2.2 Sue Oliviera

Position: Sue works as an analyst and developer in an external consulting company.

Computer Usage: Sue is a skilled computer user familiar with several scripting and programming languages. She mostly deals with relational database systems and shell scripts.

Technology Background: Sue has a vague understanding of the semantic web and related technologies. She has heard about RDF but has never applied it. She is familiar with UML modeling tools, but does not use them every day.

Framework Knowledge: Sue has heard of the m2n Framework at workshops, and experimented with it in order to find out whether it would fit her needs. However, she never used it seriously. Her knowledge is heavily based on m2n workshops.

Visualization Use: The main use case is to adapt the target ontology and to explore and change existing resources stored in the application graph. Sue also wants to model sub-graphs which follow a specific schema.

6.2.3 Scenario 1

Tom Nelson is about to adapt the target ontology for a new customer (Sue). Tom meets Sue in her office and Sue starts to explain her problem. Tom listens to the description. Anytime he identifies a concept, he creates a new class in the GraphVisualization. Then he chooses an edit, double clicks the class, enters the name and completes the process by simply hitting on the return key. Sue insists an adding some description to a class named Freelancer. Tom chooses a tool for adding the description and double clicks the class again in order to enter the information. Again the process is completed by hitting the return key. Once a class is created, Tom tries to identify the object properties and the datatype properties by asking detailed questions.

In order to create datatype properties, Tom selects another tool, double clicks the respective node and asks Sue for the name and possible values of the property. From the values Tom knows which datatype he needs to choose for the editor being provided. Sue states that there exists another relationship between the class Company and Freelancer. Tom selects a tool and clicks Freelancer. The visualization marks the node as subject. Afterwards, Tom clicks the class Company and the system immediately provides possible object properties, which are valid in the OntologyVisualization. Sue notices that the inverse relationship also exists. Tom clicks the company node and chooses the selection history in order to jump to the previous location. The viewport moves smoothly to center the class Freelancer. The process continues until both agree that the domain is complete.

6.2.4 Scenario 2

Sue Oliviera is analyzing the target ontology created with Tom in the first session. She wants to understand the classes Tom created from her description. She is looking for a special subclass of Person. However, she can not remember the exact name of the resource Tom created. Sue opens the overview window to orient herself. Since she cannot see the class person she opens the quick search, types "person" and confirms. The search tool provides a list of classes and properties containing the word. Sue selects the first entry and the viewport centers the selected element. In order to see the associated names, Sue activates two states. The first state highlights the edges, the second state additionally overlays the labels. Immediately she sees the class FreeLancer she was looking for. She sets a visual mark for orientation and begins further investigation.

6.3 Common Use Cases

The two scenarios given above lead to the following common use cases that must be supported by the GVP:

Find Resources: Most of the tasks mentioned above require a search feature. Therefore, the user should be able to find a resource by its name or by a regular expression. The system should provide a list of results. When selecting an element from the list, the viewport should center and select the element in the visualization. Moreover, all search results should be marked with a highlight color.

Overview: The GraphVisualization should provide an overview when zoomed in. As mentioned in the last chapter, most tools provide a bird's eye view. Such a view is likely to be comprehensible as it is common to most graphics applications. As discussed in the last chapter, animations can support the orientation of the user. The GraphVisualization should provide animations on zoom and pan, but let the user interrupt the animation at anytime.

Navigation Support: During the modeling process it is often required to return to previous locations. The GraphVisualization should support this in terms of a selection history. Choosing an element

from this history should restore the previous viewport position. Furthermore, it should be possible to mark resources visually using colors as visual bookmarks.

Detail Inspection: The GraphVisualization should allow the user to inspect resources in detail even when zoomed out. This can be achieved in multiple ways. First, by a magnifier which is overlaid at the position of the mouse. Second, by using excentric labels similar to those described in Fekete and Plaisant [1999]. An excentric label is located in some distance from its node and is typically connected to it with a line. In the GraphVisualization it can also include the icon of the resource. Third, the visualization might be linked to other views such that linking & brushing can be applied. For example, the OntologyVisualization should provide an alternative view using the MultiTree.

Dependency Inspection: It must be possible to see resources associated with one another at a glance. The problem results from the fact that edges often overlap when using curved edges. The GraphVisualization should use color coding to highlight such dependencies in order to support recognition.

Selection Support: When exploring or navigating through a graph, it is often required to move resources to another location. In GraphVisualization 3.5 this is not well supported since only single nodes can be selected at a time. Thus, the GraphVisualization should allow the selection of multiple resources. The visualization should also give feedback about the selected elements. If a single element is selected, the name should be shown such that it is readable on low zoom levels. If multiple elements are selected, the number of elements should be shown.

Instantiate Resource: Instantiation can be problematic when the number of producible resources is huge. For this case, a catalogue of matching resources must be available.

Improve Node Representation: The node display must be improved in order to facilitate distinction between datatype properties and literals. GraphVisualization 3.5 provides only a poor node presentation, since datatype properties and literals are not fully visible and change their order anytime the screen is refreshed. For general nodes, as used in the OntologyVisualization, the notation should follow the UML conventions. Thus, datatype properties should show their name followed by a colon and their datatype. Literals should use the schema: *name*=*value*. As names are often long URIs, tooltips should present the full content where it is too long.

Edit Resource: There must be a tool which automatically identifies the property a user wants to edit. The user double clicks an element in the visualization and starts the editing process. The GraphVisualization should provide an editor with respect to the clicked element. For example, when the user clicks the title of a node or an edge, it is most likely that the label property should be edited. Thus, the supplied editor would be a text field. When editing a datatype property, it is most likely that the user wants to change the range and the name of the property. Thus, the editor must provide different input facilities, depending on the situation.

Add Object Property: The visualization should support the creation of statements between resources while providing visual feedback. In other words, linking should be directly possible by a tool and not only by the context menu. When activated, the user should select the source and the target node in sequence. The tool must take care of the resources in question. This means, it should only allow linking for properties that would be visible and that a user is allowed to create.

Add Datatype Property, Add Literal It should be possible to create datatype properties and literals directly in the visualization. In case of a datatype property, the supplied editor must request the domain and the name. In case of a literal, the editor must request the literal and its value. The editor should only provide literals which are valid with respect to the considered resource.

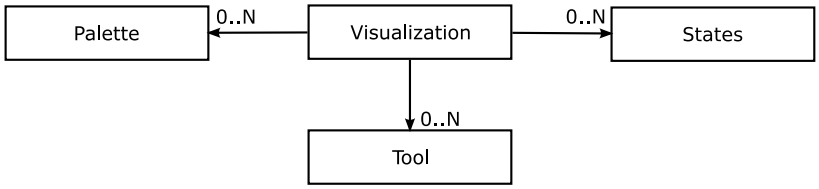


Figure 6.1: The conceptual model of the visualization.

6.4 Conceptual Model

The term conceptual model was introduced by Don Norman (Norman [1987]). A conceptual model provides a high-level description of the application and the concepts a user is faced with when performing tasks ([Johnson, 2008, p. 373]). Norman argues, that a user builds a *mental model* of the world and the system based on prior knowledge. Even though this model is incomplete and rapidly changing, it supports the human in conclusions about the world. This idea is similar to that of semantic graphs in cognitive science described in Chapter 2. Thus, in order to facilitate learning and understanding of the user interface, a model comprising a few well-known concepts is best suited (Norman [2005]).

The conceptual model for the GraphVisualization is shown in Figure 6.1 and consists of four general elements. The central concept is the Visualization which shows the actual content. The content of the visualization is always shown as a node-link diagram, whereby nodes and edges can have different appearances. More precisely, their appearance should be adaptable to the context of the visualization. The visualization can exist on its own, but can provide access to *tools*, *states*, and *palettes*.

A tool provides the means for direct manipulation of the data to be shown in the visualization. It must be activated and should help to perform the use cases *Add Datatype Property*, *Add Literal*, *Add Object Property* and *Edit Resource*, as defined in the previous section.

A state affects and reflects the presentation of the data in the visualization. For example, a state can alter the presentation of the nodes and edges to support the inspection of associated resources (use case *Detail Inspection*). These states can be used separately or in combination. On the other hand, states give feedback about the visualization. For example, the state can give feedback about the currently selected elements (use case *Selection Support*).

Finally, a palette is the most general concept. Palettes support the application of the visualization in different contexts. They can support the use cases: *Overview*, *Find Resource*, *Navigation Support*, *Instantiate Resource* and *Detail Inspection*. For example, it is possible to implement a search feature and the overview window as palettes. Moreover, palettes can provide a different kind of presentation that is linked to the central visualization to support linking & brushing (use cases *Detail Inspection and Add Object Property*).

Chapter 7

The m2n GraphVisualization Plug-In

This chapter describes the features of the completely redesigned m2n GraphVisualization Plug-In (GVP) from the perspective of a user. The GVP is available for the Swing client and can be configured in terms of the contents to be visualized and the tools available for navigation, creation, and manipulation. Standard configurations are provided for common applications, such as the OntologyVisualization, the RuleVisualization and the StarVisualization. The new GraphVisualization Plug-In has already been used to build formula trees and organizational charts.

The following sections explain the features of the interface using the OntologyVisualization as an example application of the Plug-In. The next section gives an overview of common user interface elements. Afterwards these elements are described in more detail. The last part of this chapter explains the configuration of the Plug-In using examples in N3 syntax. It should be noted that the last part is not meant to be carried out by the average user, but should show which parts of the visualization can be configured in general.

7.1 Common Interface Elements

The GraphVisualization Plug-In visualizes and manipulates selected parts of the application graph. Nevertheless, the basic user interface components remain the same and fit the conceptual model shown in Figure 6.1. Figure 7.1 shows an example configuration of the GraphVisualization for target ontology visualization. The component is subdivided into three main parts: a Toolbar on top, the Main View in the center and a Statebar at the bottom. The availability of the Toolbar and the Statebar is subject to the component's configuration.

The Main View is always available and shows the graph in an infinite two-dimensional plane. In the OntologyVisualization, nodes represent the classes of the ontology and edges represent object properties of these classes. Angular edges show a direct subclass relationship. The viewport can be moved by dragging the mouse while holding the right mouse button or by using the scrollbars. In order to increase or decrease the zoom, the mouse wheel or an overview window, called the *Navigator,* is provided. When using the mouse wheel, the zoom follows the current mouse position. All elements in the visualization can be selected by left clicking or with rectangular selection.

In comparison to the old visualization, the display of nodes has been improved in multiple ways. As before, the nodes show the class label but also a selected set of datatype properties and literals. Additionally, an icon is shown next to the title which represents the value of the property `owlx:classIcon`. Furthermore, the new representation applies a clear order to datatype properties and literals using the UML-style schema. Datatype properties always show their name, followed by a colon and their range. Literals are shown as name-value pairs using the schema: "name=value". The alternating colors in each line provide improved readability. Since a node is often too small to display its whole URI, a tooltip

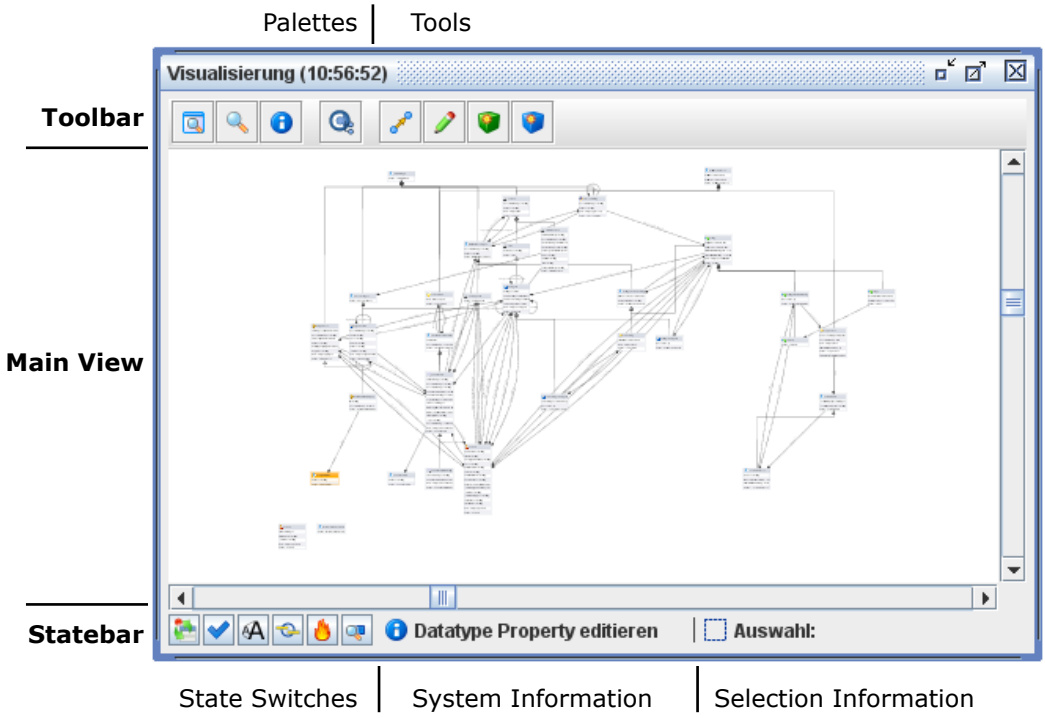


Figure 7.1: An example for the interface of the GraphVisualization when displaying target ontology. The availability of tools, state switches and palettes is subject to the component configuration.

becomes visible when the mouse resides over a node. Figure 7.2(a) shows the old and Figure 7.2(b) the new display of general nodes used in the OntologyVisualization.

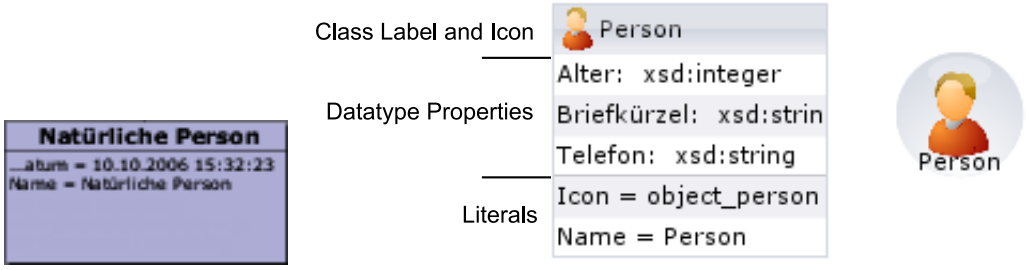
Nodes support *visual landmarks* using different colors. The user can visually mark a node or a set of nodes by changing their background color. This is helpful in huge graphs and in cases where a node appears twice.

7.2 Toolbar Elements

The toolbar contains palettes and tools for manipulating the resources shown in the Main View. The first button group in Figure 7.1 gives access to palettes having fixed functionality like the Navigator and the QuickSearch. The second group hosts palettes whose functionality is configurable through component configuration. The third group contains tools for manipulating currently shown resources. The availability of these tools is also part of component configuration.

7.2.1 Navigator

The Navigator shown in Figure 7.3 provides a bird's eye view of the entire graph. In order to accelerate the rendering process, nodes are shown with a lower level of detail, such that only their bounding boxes are drawn when the palette is small. Dragging the yellow rectangle in the Navigator changes the viewport position in the Main View. It is also possible to click a location in the Navigator panel. In that case the viewport of the Main View is animated, such that the clicked position becomes centered. In addition, the Navigator checks whether this position contains a resource and selects it. To ease this indirect selection from the Navigator tooltips are show as well.



(a) General Node **(b)** General Node and Icon Node

Figure 7.2: A comparison of old and new node representations. (a) shows the old version of *general node* representation having a fixed height and showing predefined literals or datatype properties in no specific order. (b) shows the new general node and icon node representation available in the OntologyVisualization and other visualizations. The new general node applies a strict order to a configurable set of datatype properties and literals. The icon node representation provides a condensed view.

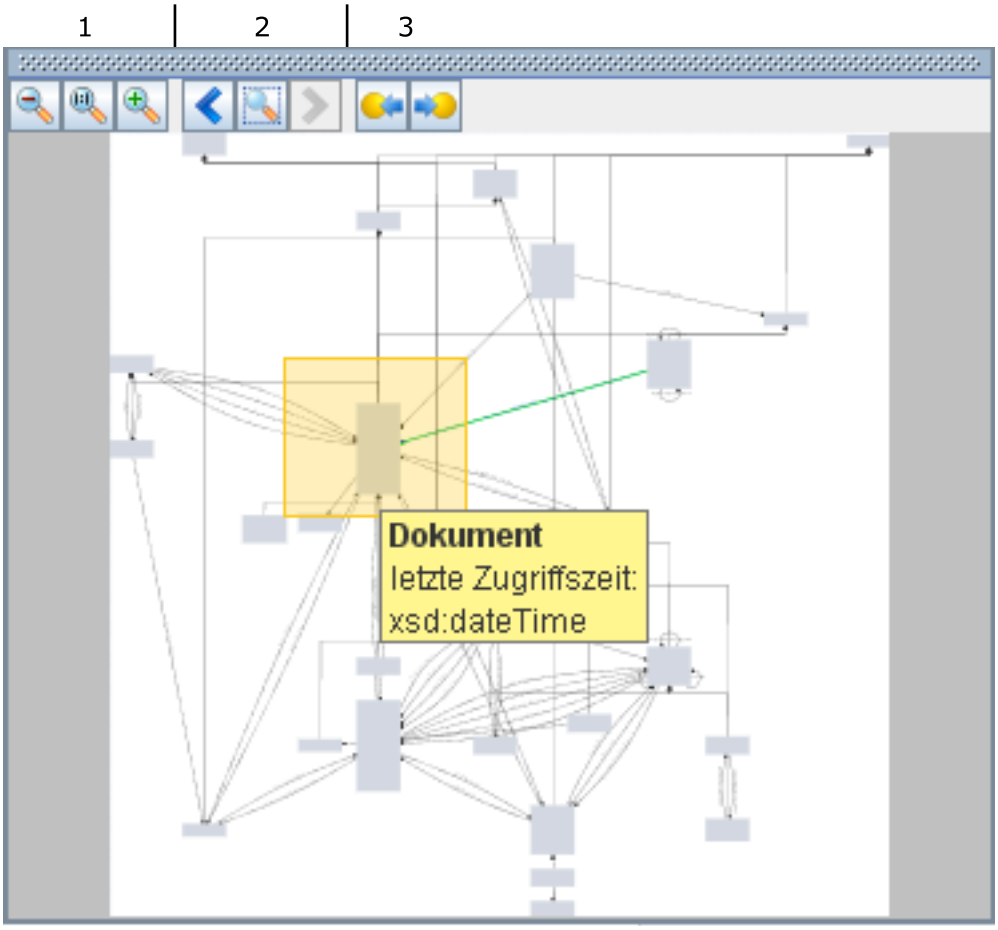


Figure 7.3: The Navigator provides a bird's eye view of the entire graph. Tooltips simplify indirect selection of resources. Buttons at top give quick access to zoom (1), to a selection history (2) and support quick navigation between the subject and the object of selected edges (3).

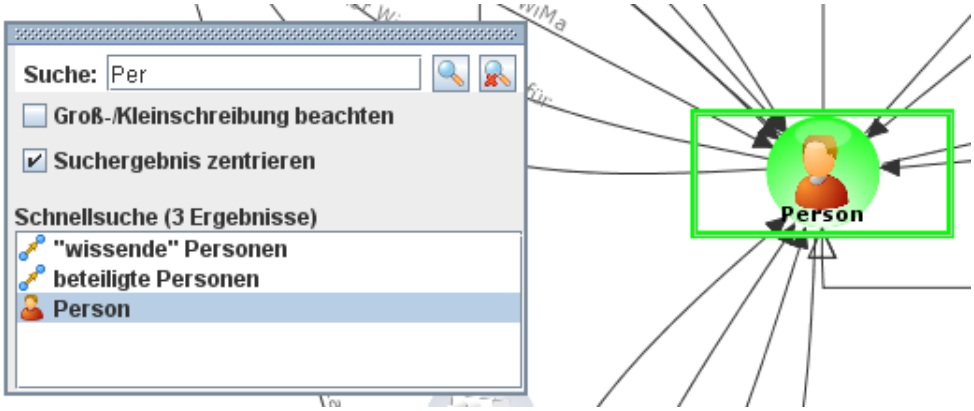


Figure 7.4: The QuickSearch palette queries the properties which are shown as label. Results are presented as alphabetically ordered list. Found nodes are marked with a green background in the Main View. Selections in the list center the viewport of the Main View accordingly.

As shown in Figure 7.3, the Navigator provides further functionality through the action buttons at the top. The first group provides the means to decrease, reset, and increase the zoom of the Main View. A reset of the zoom to 100% is animated in order to facilitate orientation. The second group of action buttons provides access to the selection history. This means, a click on these buttons centers the viewport on the previously or the currently selected element. This repositioning is animated as well. The selection history works as in a web browser and stores the previous ten selections of single resources. If multiple resources are selected, only the first is stored in the list. The third group of buttons can be used to navigate to the source or the target node of an edge. They only become active when an edge is selected in the Main View. This functionality aids navigation in large graphs where edges overlap, for example in the visualization of rules.

7.2.2 QuickSearch

The second button in the toolbar gives access to the QuickSearch palette. The QuickSearch can query the properties which are used as a label in the visible elements. By default, the search request is transformed into a regular expression such that the result string only needs to contain the search term, but must not exactly match it. However, more advanced users can use regular expressions as described in Zakhour et al. [2009].

As shown in Figure 7.4, the results of a search are presented in an alphabetically ordered list. Additionally the background color of found nodes are set to green in the Main View. Selecting a result from the result list animates the viewport of the Main View in order to center the selected element. Additionally, the element is selected. Since the animation can be annoying when the graph is viewed at small zoom levels, it can be suppressed by unchecking the "Suchergebnis zentrieren" control.

7.2.3 Configurable Palettes

The second group of buttons in the toolbar controls palettes whose functionality depends on the component configuration. For example, in case of the OntologyVisualization a single button exists which toggles a palette containing a MultiTree. The tree shows the contents of the ontology in an additional view which might be easier to understand in some situations. Moreover, the tree is *synchronized* (linked) with the visualization. Selections in the visualization lead to selections in the tree, if the respective resource has already been expanded in the tree. In turn, selections in the tree lead to selections and an-

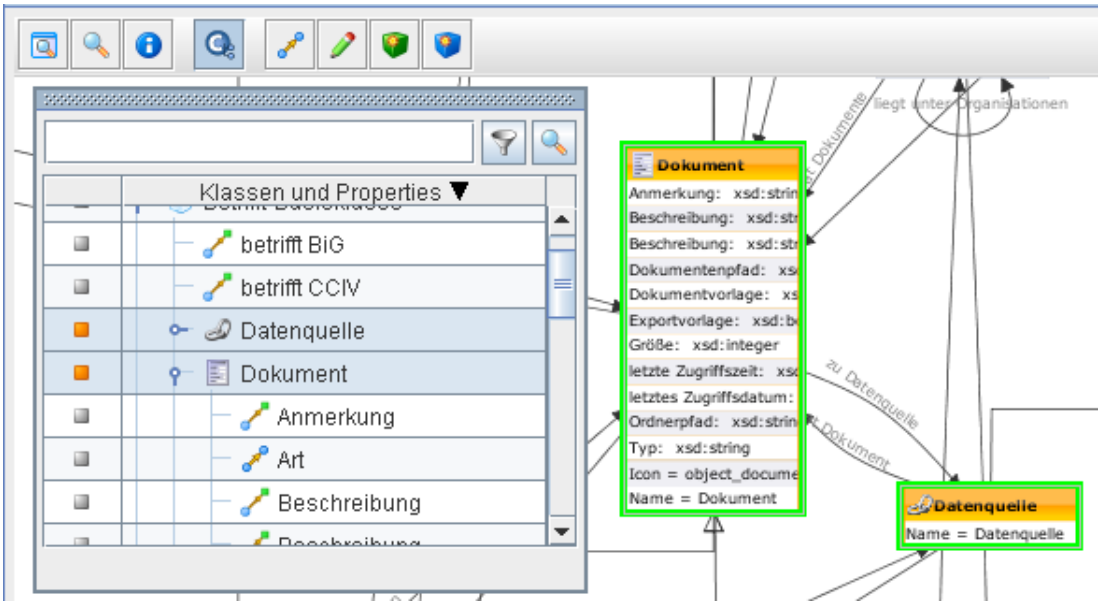


Figure 7.5: The visualization can be linked with different widgets. The example shows a configured palette which shows the target ontology in a MultiTree widget. Selections and *visual landmarks* from the palette are synchronized.

imations in the Main View, if the respective resource is visible. Beside selections, visual landmarks of nodes are synchronized as well.

Figure 7.5 shows an example of this behavior in the OntologyVisualization. The resources Datenquelle and Dokument had been selected and visually marked in orange in the Main View. As a result they become selected and marked in the MultiTree as well.

The described linking & brushing behavior between the MultiTree and the Main View is specific to the OntologyVisualization. This means, it need not always be available. For example, the RuleVisualization provides a palette which contains rule blocks instead. This palette is not synchronized but serves as a container for resources which can be instantiated for creating a rule. Therefore, a user would drag a block from the palette and drop it into the Main View. In general, the number of toggle buttons and the functionality of the associated palettes depends on the application. It would also be possible to include a second GraphVisualization if required.

7.2.4 LinkTool

The rightmost part of the toolbar hosts a set of tools for manipulating the resources in the Main View. The LinkTool is used to create a single or multiple statements between two resources. When activated, the creation is triggered in a two-step process as shown in Figure 7.6. In the first step, the user selects the subject node. To indicate that the selection succeeded, the node is marked pink. The second step is to find the resource which should serve as the object of the statement. When the object node is selected, a pop up menu appears and the user can select the type of statement. The available types depend on the configuration of the visualization. Figure 7.6 shows a typical example for the link menu in the OntologyVisualization. The RuleVisualization would contain properties with respect to the underlying rule schema instead.

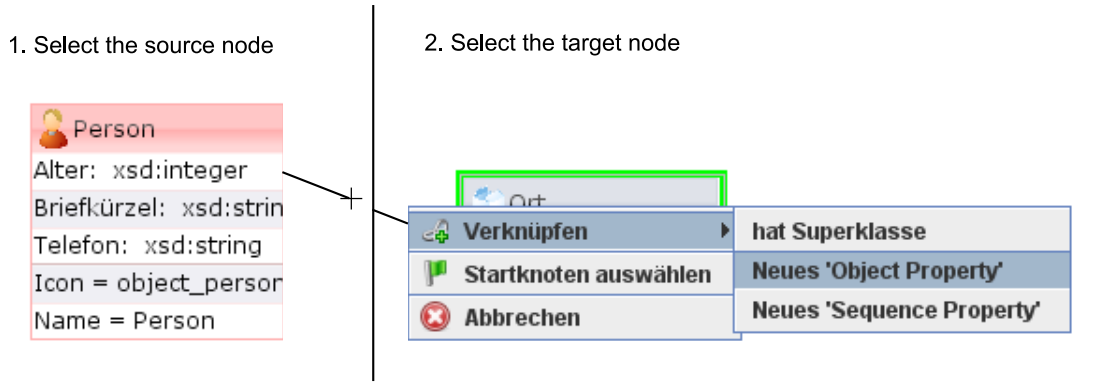


Figure 7.6: The LinkTool is used in two steps. First the source and later the target node is selected. The source node appears in pink. The link menu contains options dependent on the configuration.

7.2.5 QuickEdit

The QuickEdit tool provides a mean to change the shown datatype properties and literals. To start the editing process the resource is be double-clicked. Depending on the clicked resource, the QuickEdit tool determines what kind of editor is needed. For example, a double-click on the title shows a simple text editor where the user can change the literal associated with the label property. In contrast, when double-clicking a datatype property, the editor provides the means to change the name and the range. For the sake of simplicity this editor only contains the most common datatypes defined in the XML schema definition like `xsd:integer` or `xsd:string`.

It is currently not possible to add custom datatypes without using the DetailView. Figure 7.7 shows the editors for datatype property and literal editing. Note that is also possible to edit edges. In this case it is assumed that the label should be changed. Thus, a literal editor as shown Figure 7.7(a) would appear.

7.2.6 Creating Datatype Properties and Literals

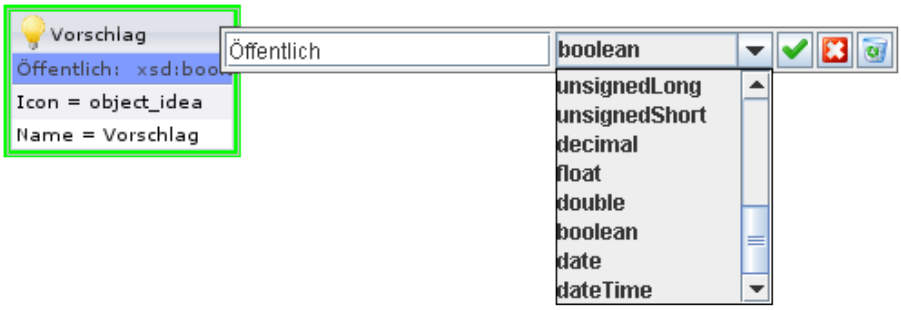
A common use case is the creation of datatype properties and literals. As with the editing of these properties, creation requires different information to be entered by the user. Hence, an appropriate editor must be supplied depending on the type of statement that should be created.

Given the case that the user wants to create a datatype property, the editor is identical to the QuickEdit editor shown in Figure 7.7(b). The delete button is missing since the statement is not part of the application graph yet. In case a literal should be added the editor provides a combo box and an editor. The combo box contains the literals defined in the schema and the editor is used to enter the value of the literal. Currently, only a text field is available to edit the values. Thus, any value must then be converted from text to the typed literal and vice versa. However, this is a point for improvement since it is sometimes required to use a strict format such as date values. Therefore, further editors will be added in later revisions of the GVP.

7.3 Statebar Elements

The bottom part of Figure 7.1 shows the statebar where the user is able to activate certain visualization states and access context information. The field *system information* is used to show state messages from the system. For example, when something is dropped into the visualization in a drag & drop operation and the user is not allowed to do so, the field shows an error message. The same applies if linking of two resources is forbidden or simply not possible in the current configuration. The field *selection information*

(a) Literal Editor



(b) Datatype Property Editor

Figure 7.7: The QuickEdit tool supports the modification of literals (a) and datatype properties (b). Depending on the selected element, an appropriate editor is opened. The tool also works on edges in the visualization.

shows information about currently selected resources. If only a single resource is selected, the label of the selected resource is shown. If more than a single resource is selected, the number of resources is displayed.

The buttons in the statebar are used to change the appearance of the visualization and provide the means to enable further "investigation states". Some of these states can be combined as explained in the next sections.

7.3.1 Icon Switch

The *icon switch* toggles the appearance of certain nodes between the general representation and icon representation shown in Figure 7.2(b). The purpose of the switch is to provide more overview when nodes have a huge number of attributes. However, its availability depends on the configuration of the GraphVisualization. For example, in the RuleVisualization it would not make much sense to alter the appearance of parameter nodes or conditional blocks since they have a fixed meaning to the user.

7.3.2 Synchronization Switch

The second button is called the *synchronization switch* and is only available in case the visualization is synchronized with another view. The switch toggles the animation on synchronization events. When turned off the viewport will not be animated when selecting elements in a *synchronized component* (see Section 7.2.3). This is helpful when watching a graph at low zoom levels and animation becomes more distracting than supportive.

7.3.3 Title Scale Switch

The *title scale switch* toggles the magnification of node labels. If activated, all titles are scaled inversely to the current zoom factor. The bounding box is respected in order to avoid overlapping titles. Figure 7.8

(a) off **(b)** on

Figure 7.8: The title scale switch toggles the scaling of the node label within the bounding box of the node. When turned on, the user is able to read part of the title at low zoom levels.

shows the effect of the scaling turned on and turned off. Inverse scaling is only applied at zoom levels smaller than 100%.

7.3.4 Inspector

The title scaling described above is only useful for a limited magnification range since the label becomes much larger than the bounding box of a node for low zoom levels. On the other hand, it is not feasible to extend the bounds of the nodes in order to fit the whole label since they are potentially quite long. The major question is: what part of the label should be shown, and how many labels should be visible at once?

The *Inspector* tries to address this issue by displaying the titles of certain nodes and edges as an overlay. It provides two modes of operation. First, if nothing is selected, all node titles in an area around the mouse cursor are shown. This *sensitive area* is shown as a blue circle in Figure 7.9(a). The diameter of the circle can be set using the mouse wheel while pressing the control key. When moving the mouse, the Inspector computes the intersected nodes and displays their labels. The labels are ordered with respect to distance from the cursor tip. The node closest to the cursor is marked with an additional black border. This border also indicates which node would be selected when clicking the left mouse button. In this way, the Inspector can be used to select elements precisely at low zoom levels.

In the second mode, if a node or an edge is selected, only the title of the selected element is overlaid as shown in Figure 7.9(b). The underlying assumption is that this node is the most important one for the user. Therefore, the label is allowed to overlay anything else.

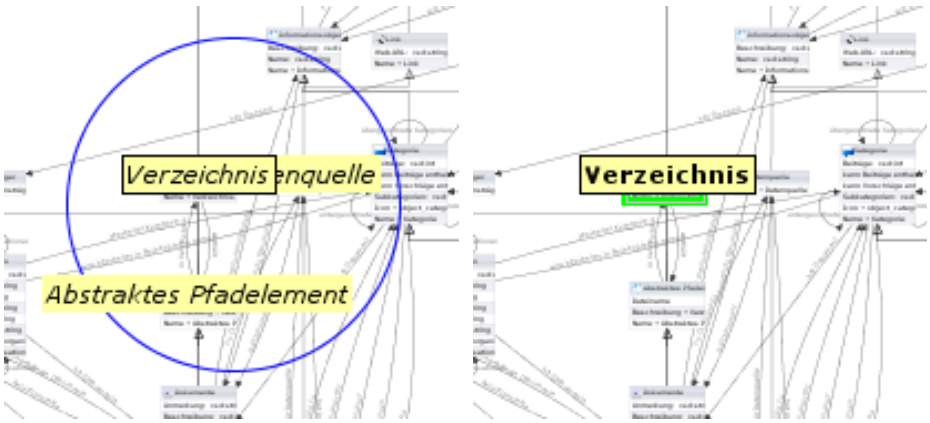
When zooming in to a certain degree, the overlay disappears. This avoids interuption caused by activating and deactivating the Inspector.

7.3.5 Search Light

The *Search Light* serves to highlight associated resources using color, as shown in Figure 7.10. If a node is selected all associated object properties and objects are shown framed in yellow. An important feature is that the Search Light and the Inspector can be combined to gain more insight. When selecting a node with both states enabled, the titles of the associated nodes are overlaid as well (Figure 7.11). The same applies for edges. When an edge is selected, its associated nodes are highlighted and the labels of the node and the edge are overlaid.

7.3.6 Magnifier Switch

The last switch provided in the statebar is the *magnifier switch*. When activated, a magnification window is overlaid at the current mouse position. In order to simplify the selection of nodes, the background of the magnified area is not fully opaque as shown in Figure 7.12. As with the Inspector, the magnification window disappears when zooming past a certain level to avoid interuptions of work.



(a) No Selected Element **(b)** Selected Node

Figure 7.9: The Inspector overlays the titles for nodes and edges. If no nodes or edges are selected, labels of nodes within a resizable sensitive area (blue circle) are overlaid and sorted with respect to their distance to the cursor tip (a). The dark framed node in (a) would be selected on mouse click and the visualization would appear as shown in (b).

7.4 Content Configuration

The content configuration of the visualization describes *what* should be shown in the Main View of the component and *how* it is represented. The configuration also provides properties which define whether the LinkTool, the DatatypeProperty Creator or the LiteralCreator are available. Listing 7.1 shows an excerpt of the configuration used in the OntologyVisualization. A configuration includes three main parts: *node configuration*, *edge configuration* and *literal configuration*.

The node configuration determines the resources which should appear as nodes and how they are rendered. For example, it is possible to define that all nodes having a certain property should be visible in the visualization. This, for example, is required for visualizing the resources of the target ontology. All nodes in this visualization are subclasses of the class owlx:Zielontologie. Listing 7.1 (lines 19–24) shows how node configuration would look in this case. The resource `ex:OntologyNodes` is declared as config:PropertyNode. Two object properties can be added to this resource. The object property `config:nodeProperty` defines all predicates a resource must have in order to appear in the visualization. The property `config:nodeResource` declares the objects that is associated to this property. Thus, in order to appear in the OntologyVisualization, the property `config:nodeProperty` contains the value rdfs:subClassOf and `config:nodeResource` contains the value owlx:Zielontologie.

The node configuration described above is only one example to show how nodes are selected in general from the application graph. Further node configuration resources exist and the content configuration can declare multiple such configurations. Furthermore, the node configuration sets the renderer for the nodes which match its criteria. In general, a renderer defines how the node is going to be drawn in the visualization. In line 24 of Listing 7.1, the renderer for general nodes is assigned. Further renderers are available for visualizing rule elements.

The principles described for node configuration also apply to edge and literal configurations. Listing 7.1 provides an example in lines 27–32, where the subclass edge is configured. The configuration defines a resource which has the type config:SelectEdge. This resource requires a property called `config:edgeProperty`. The effect of this property is to select all predicates from the application graph whose RDF type equals the object given in this statement. Thus, for the example, all properties having the RDF type owlx:directSubClassOf appear in the visualization. Like in the the node configuration, their appearance is determined by a renderer associated in line 28.

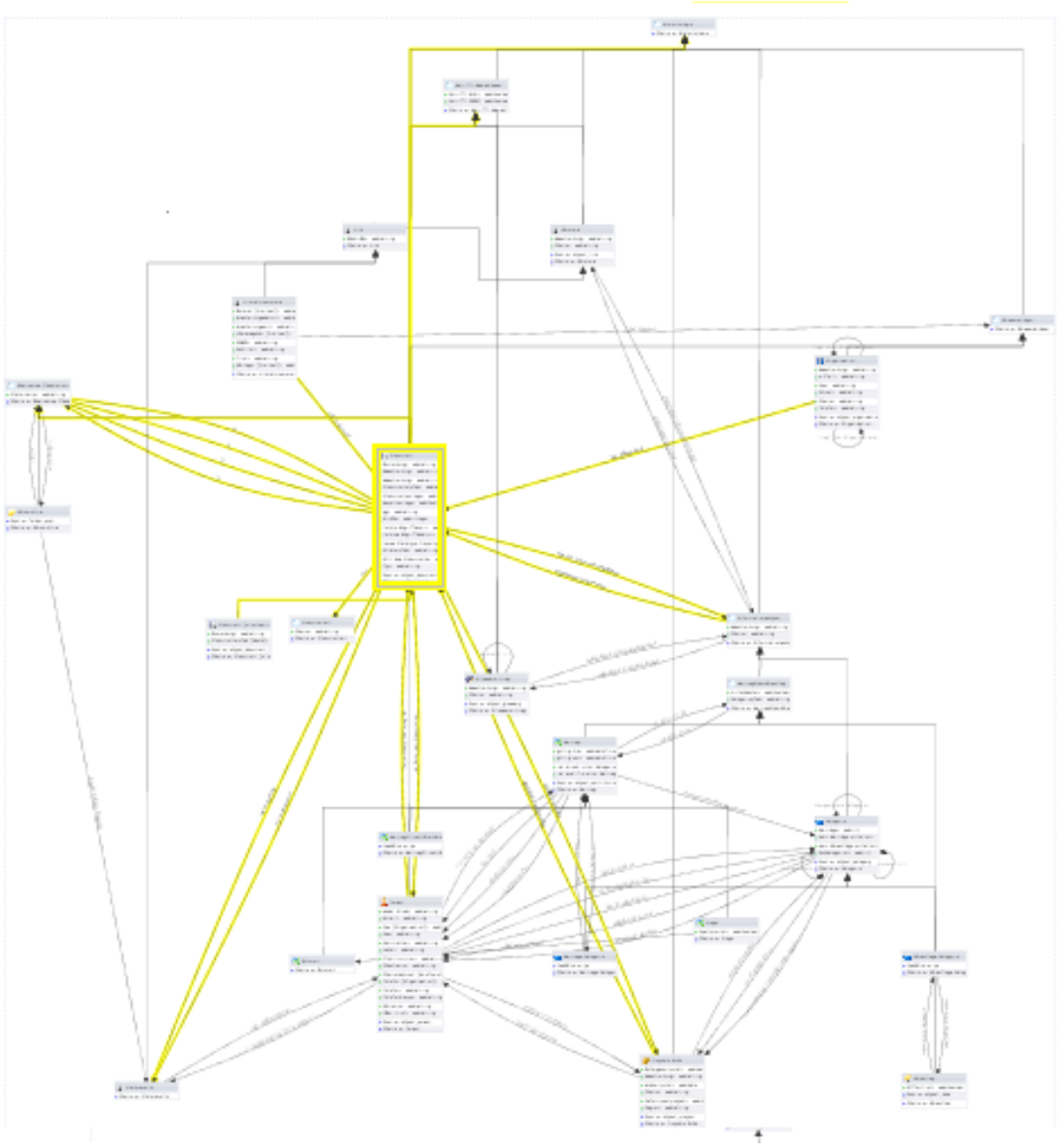


Figure 7.10: The Search Light uses color coding to highlight associated nodes and edges, making it easier to recognize associated resources.

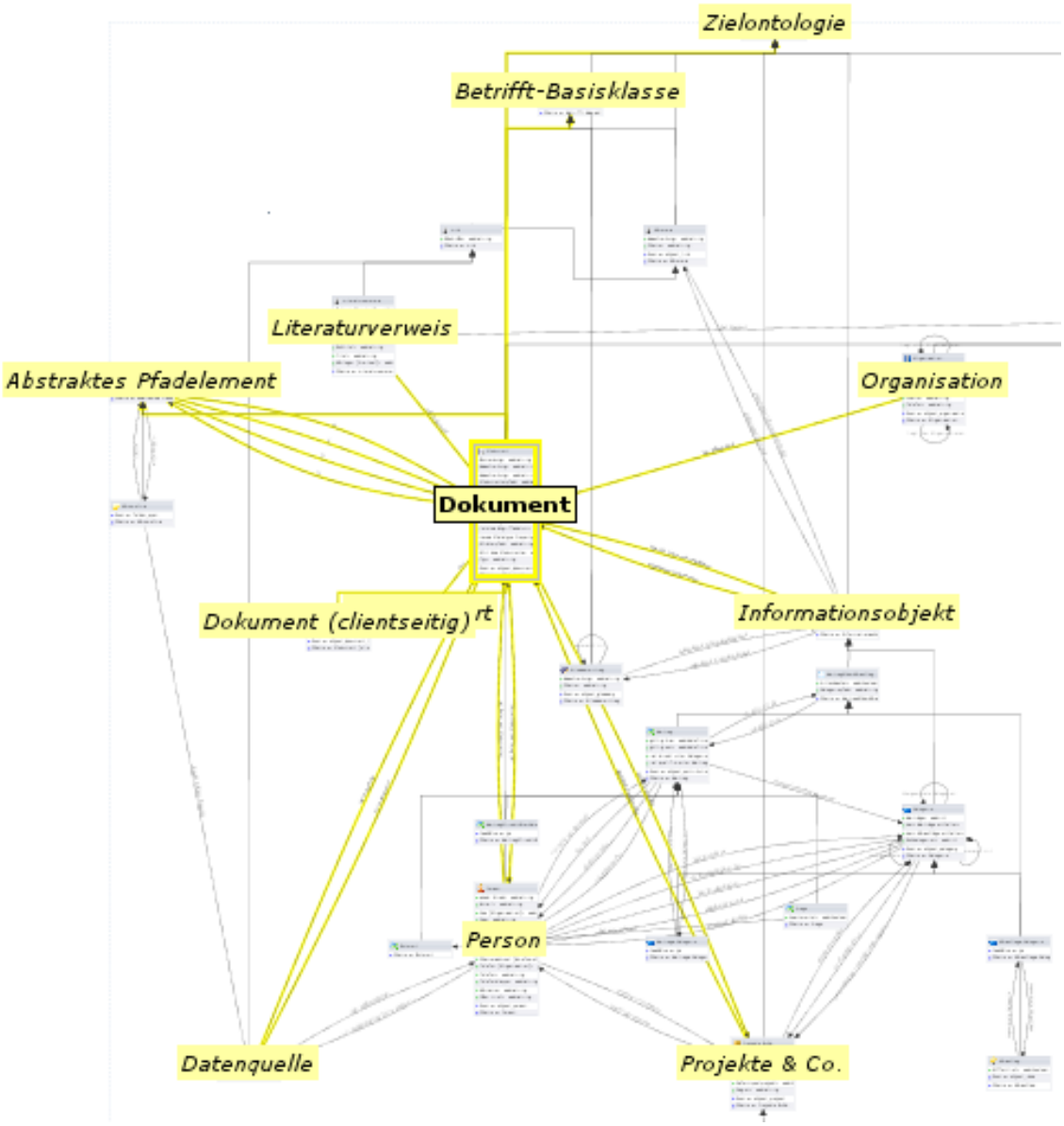


Figure 7.11: When using both the Search Light and the Inspector, the titles of associated nodes are overlaid as well. The combination of these states also works for edges.

```
1  # The resource that serves as content configuration
2  ex:OntologyVisualization a config:Class ;
3      # the title of this resource when it is viewed in another component
4      config:title "Ontology Visualization Configuration" ;
5      # LinkTool is available
6      config:provideLinkSupport "true"^^xsd:boolean ;
7      # QuickEdit is available
8      config:provideEditSupport "true"^^xsd:boolean ;
9      # Literal creation is possible
10     config:provideCreateLiteralSupport "true"^^xsd:boolean ;
11     # Datatype creation is possible
12     config:provideCreateDatatypePropertySupport "true"^^xsd:boolean ;
13     # Resources configurations
14     config:node ex:OntologyNodes ;
15     config:edge ex:ObjectProperties, ex:SubClassEdge;
16     config:literal ex:AllLiterals, ex:DatatypeProperties ;
17 .
18 # node configuration
19 ex:OntologyNodes a config:PropertyNode ;
20     config:includeResources "false"^^xsd:boolean ;
21     config:nodeName         "Target ontology classes"^^xsd:string@en ;
22     config:nodeProperty     rdfs:subClassOf ;
23     config:nodeResource     owlx:Zielontologie ;
24     config:nodeVisual       node:DefaultNode ;
25 .
26 # edge configuration for subclass edges
27 ex:SubClassEdge a config:SelectEdge ;
28     config:edgeVisual       edge:ManhattanEdge ;
29     config:edgeProperty     owlx:directSubClassOf ;
30     config:edgeUserChange   "false"^^xsd:boolean ;
31     config:edgeVisible      "true"^^xsd:boolean;
32     config:edgeName         "Vererbungs–Kanten"@de .
33 # ...
```

Listing 7.1: An excerpt from the content configuration for the OntologyVisualization. All resources being a subclass of the target ontology should appear as nodes (lines 19–24). Furthermore, direct subclass relationships should be shown as rectangular edges (line 28).

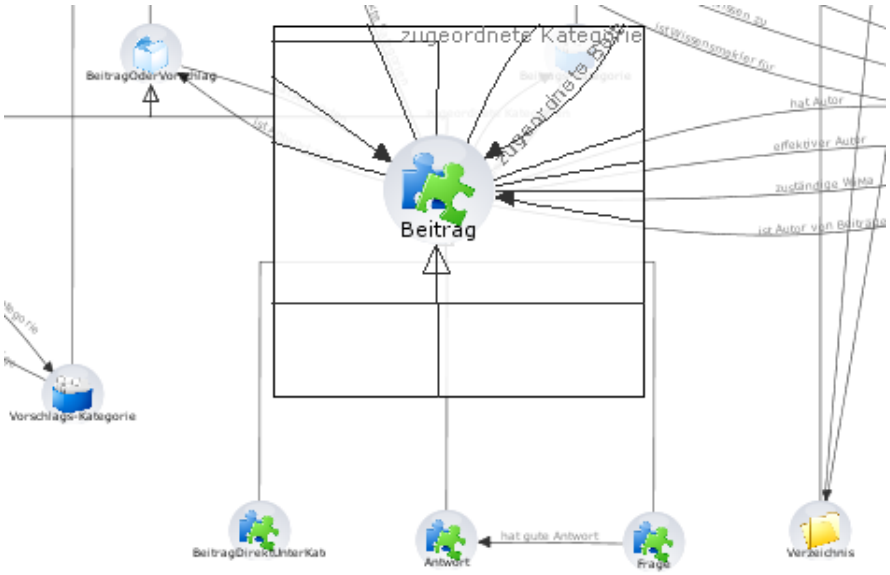


Figure 7.12: The Magnifier as a layer-based focus & context technique. The background is semi-transparent to aid selection of elements.

7.5 Component Configuration

The second part of the configuration concerns the description of the visualization as a visual component of the GUI Plug-In. Listing 7.2 shows an example of the component definition with a single additional palette. The actual content of the palette is omitted since it requires definition of a container for its layout.

In lines 1–9 of Listing 7.2, a resource ex:GraphVisualization of the type gvc:GraphVisualization is defined. The type gvc:GraphVisualization refers to a visual component which had been implemented to use the GraphVisualization as part of the GUI Plug-In. Some details of this implementation are described in Chapter 8.

In Listing 7.2, the resource ex:GraphVisualization defines three important properties: The datatype properties `gv:showNavigator` and `gv:showSearchBox` are used to control the availability of the buttons for the Navigator and the QuickSearch. The object property `gv:availablePalettes` adds an RDF sequence resource which contains a single element: ex:GraphVisualization.Palette1. As shown in line 11, this resource refers to a visual component of type vc:window which is interpreted as the definition of a palette. This means, when loading the GraphVisualization, a toggle button is created for each resource in the sequence having the type vc:window. The actual content of this window does not matter so that any complex functionality can be added. The created button is configured with respect to the properties of the window resource. For example, the title "Ontology Tree" appears as tooltip of the button and the literal value of the property `window:icon` appears as icon.

7.6 Summary

The GraphVisualization Plug-In provides a configurable component which can be used to visualize selected parts of an application graph. Tools are provided which allow the user to navigate quickly through the graph. States like the Inspector or the Search Light help to explore dependencies between resources. Palettes provide alternative representations of the graph. They can be linked to the visualization or serve as a container for other resources which can be dropped into the Main View.

```
# visualization component
ex:GraphVisualization   a   gvc:GraphVisualization ;
    gvv:showNavigator   "true"^^xsd:boolean ;
    gv:showSearchBox    "true"^^xsd:boolean ;
    gv:availablePalettes [
        a   rdf:Seq;
          rdf:_1    <ex:GraphVisualization.Palette1> ;
    ] ;
.
# additional palette definition
ex:GraphVisualization.Palette1 a  vc:window ;
    window:internalWindow "true"^^xsd:boolean ;
    window:icon           "action_show.png"^^xsd:string ;
    window:resizable      "true"^^xsd:boolean ;
    window:visible        "true"^^xsd:boolean ;
    window:width          "230"^^xsd:int;
    window:height         "400"^^xsd:int;
    window:maximize       "false"^^xsd:boolean ;
    window:title          "Ontology Tree"^^xsd:string@en;
    window:content [
        # palette content defintion...
    ] ;
.
```

Listing 7.2: Excerpt of the component configuration of the GraphVisualization including the definition of a single additional palette. Palettes are associated by the property availablePalettes. For each resource in the sequence a ToggleButton is created in the Toolbar of the visualization. These resources must have the RDF type of a window component.

In comparison to the old visualization, the manipulation of existing resources is significantly simpler. While the old Plug-In required the use of multiple DetailViews and MultiTrees, the new Plug-In supports direct manipulation using simple tools which make meaningful assumptions.

The tools currently provided do only support common use cases and some features will be added in later versions. For example, there is no support for adding cardinalities to object properties without using the DetailView. However, since the conceptual model serves as a framework for extending the visualization, such features can be added easily.

In contrast to the semantic graph visualizations discussed in Chapter 5, there is no direct way of loading and saving a semantic graph. Instead a configuration graph must be supplied which defines an excerpt of the application graph. Any operation is directly applied to this graph and has a direct effect on the application. The creation of the configuration graph requires fundamental knowledge of the available RDF configuration types. Thus, these steps are not meant to be performed by the average user. Finally, it should be noted that the Plug-In does not visualize semantic graphs for its own sake, but supports dynamic changes to a running m2n installation.

Chapter 8

Implementation Details

This chapter describes some selected details of the implementation. Since the GraphVisualization Plug-In is built on top of the m2n GUI Plug-In and JGraph (Benson [2006]), the first section emphasizes the extensions made to JGraph and describes how the new Plug-In is integrated into the GUI Plug-In. The remaining sections show some of the obstacles which had to be solved during the implementation, in particular the adapted viewport, caching renderer support, the Inspector, and adapted editing support.

Before going into further detail, it should be explained why JGraph was chosen to rewrite the old GraphVisualization Plug-In. After a code review, the old visualization turned out to be not extensible enough. Its original purpose was the visualization of the graph as a StarVisualization and efforts had been made to support modeling through the context menu. However, the presentation and interaction code was tightly coupled, resulting in a small number of huge classes. Changing one part required changing many other parts. Thus, it was decided to look for a framework which provides a more general approach.

JGraph was chosen with respect to the following characteristics. First of all, JGraph is a Swing component which makes the integration into the GUI Plug-In easy. Second, JGraph provides an extensive manual, very good code documentation, and a stable user community. This results from the fact that the component is under development since 2001. Third, JGraph provides features like inline editing, grouping, and selection support for multiple elements. The basic infrastructure is easily adaptable while always having a working version for delivering to a customer. Fourth, JGraph seemed to be the fastest general graph drawing component available. As opposed to other visualization libraries like JUNG (`http://jung.sourceforge.net/`) or ZUI toolkits like Piccolo (`http://www.piccolo2d.org/`), the visualization is fast, even if more than 1000 nodes and edges are shown. Lastly, JGraph can be used under the Mozilla Public License or the GNU Lesser Public License, which is compatible with the requirements of m2n.

8.1 Basic Structure of the GraphVisualization Plug-In

In order to understand how the GraphVisualization Plug-In is built on top of JGraph and integrated into the GUI Plug-In, the general concepts of JGraph must be explained first. For this purpose, Figure 8.1 is a good starting point. JGraph is a Swing component whose basic design follows the Model View Controller (MVC) pattern (Agrawala and Heer [2006]). In this design, the model is defined by the interface called GraphModel whose default implementation is the class DefaultGraphModel. The model stores the contents of a graph and provides the functions for manipulating the data stored about the nodes and edges. The data is stored in so-called graph cells. By default a cell can hold information about fonts, colors, and a reference to a *user object*. The adapted implementation exploits this user object to store a data structure which references RDF resources of the application graph.

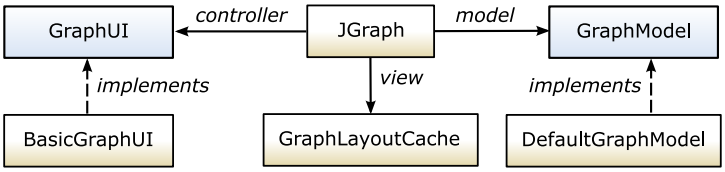


Figure 8.1: A simplified class diagram of JGraph emphasizing the Model View Controller (MVC) pattern. Yellow boxes denote classes and blue boxes denote interfaces.

The second important class in Figure 8.1 is the GraphLayoutCache. The cache maps the cells stored in the model to CellViews. Among other things the view associates a cell with a *renderer*, a *handle*, and an *editor*. The editor can be used to change the user object of the cell. The default implementation assumes that this object is a simple string. The handle is used to move the visual representation, and the renderer is used to draw the view. This decoupling of the painting code into an extra object is part of the *flyweight pattern*. That is, any time the graph is drawn, the GraphUI iterates over all views and configures a special component to perform the actual painting. The advantage of this approach is a low memory footprint and simple extensibility. However, there are also some serious issues which are further discussed in Section 8.3 and Section 8.4.

The last important class is the BasicGraphUI which implements the controller part of the MVC pattern. This class consists of a number of event handlers which define how the graph reacts to mouse and keyboard events. Moreover, the UI is responsible for controlling the rendering of the graph.

The GraphVisualization extends the class visual component from the m2n GUI Plug-In and the main JGraph classes discussed above in order to implement the requested features. Figure 8.2 shows a simplified class diagram of the most important classes, which will be referred to in the following sections. The left part of the figure shows the classes which implement the interface for the m2n GUI Plug-In. Thereby, the class GraphVisualization corresponds to a resource in the application graph having the RDF type gv:GraphVisualization. In other words, it provides access to the properties of the resource defined in the component configuration described in Section 7.5.

The class GraphVisualizationPanel reads these properties when loading a visualization and provides the implementation of the GraphVisualization for the Swing client. The panel is a container for the Main View, the Toolbar, and the Statebar. When loading a visualization the panel queries properties from GraphVisualization and instantiates classes like the GraphScrollPane, the DefaultGraph, the Toolbar, and its palettes. Moreover, it instantiates an object for the content configuration resource and invokes the loading process of the resources from the application graph. Lastly, this class implements the linking & brushing functionality to synchronize the selection and visual landmarks between with other widgets of the GUI Plug-In.

The right part of Figure 8.2 shows classes extended from JGraph. The next sections discuss some of them to show the obstacles which occurred during the implementation.

8.2 The GraphScrollPane and GraphViewport

In order to display graphs larger than the screen, JGraph is designed to work together with JScrollPane and JViewport provided by Swing. Thereby, JViewport represents a camera looking at the graph and JScrollPane provides the means to move this camera within its bounding box using scrollbars. The setup of these two components is shown in Figure 8.3(a). This simple setup is well-suited to relatively simple Swing components like JTree or JTable. These components usually have a much simpler layout than general graphs and do not support moving the contents freely into arbitrary positions. Furthermore, zooming is not provided in most Swing components. Such assumptions allow the JViewport to be implemented

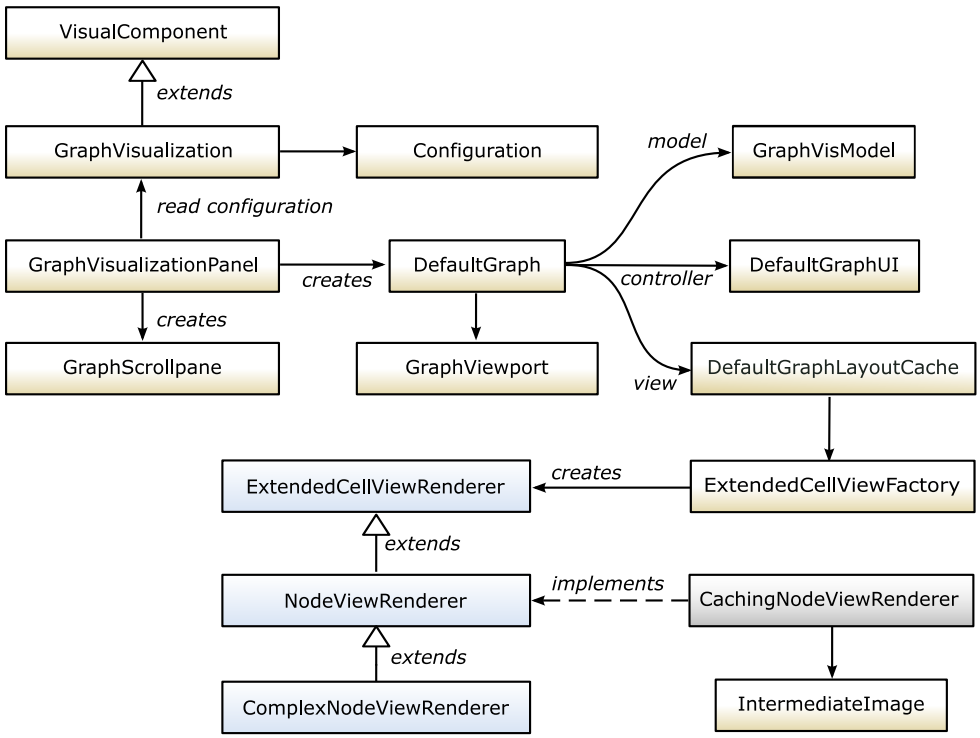


Figure 8.2: A simplified class diagram of the m2n GraphVisualization Plug-In (GVP). Yellow boxes denote classes, blue boxes denote interfaces, white boxes denote abstract classes.

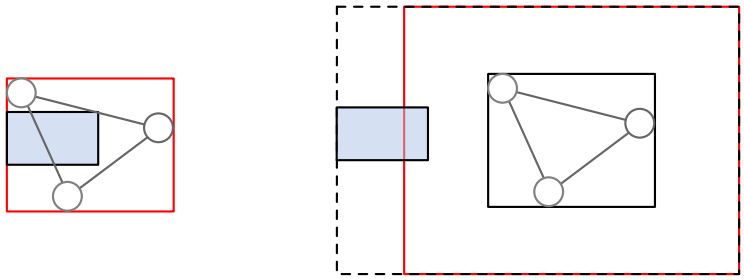
with certain rendering optimizations. However, for the display of a general graph, the following issues occurred:

1. JViewport does not allow negative coordinates as used by the old GraphVisualization Plug-In. As a result, the coordinates of all resources opened in the old visualization became invalid.

2. JViewport can only be moved within the bounding box of the client area. Thus, elements shown close to the border can never be centered in the viewport. When palettes were shown, elements were often occluded. This made the Navigator unusable, since the user was forced to reposition the palette or move the nodes.

3. Dynamically resizing the client area caused the viewport to jump around and led to massive painting errors. This happend especially in the StarVisualization when a node was moved close to the left border. Expanding this node led to the problem that its children were placed in negative coordinates. Additionally the client area expanded. This caused two updates whose order of occurrence was nondeterministic. One event resized the client area and the other event updated the scrollbars of the JScrollPane. As a result the position of the viewport was only sometimes correct the with respect to the expansion.

In order to avoid these problems, the classes GraphScrollPane and GraphViewport were implemented. The GraphScrollPane provides an infinite two-dimensional plane which is automatically resized such that it fits the whole graph including a margin. Figure 8.3(b) shows a schematic view of the the viewport setup used in the GraphVisualization. The center shows the actual graph and its bounding box. This bounding box is extended by half its height and width. The resulting rectangle defines the minimum scrollable area. Within this area, the viewport can be moved using the scrollbars. However, in contrast to JViewport, the GraphViewport can be moved outside of the minimal client area by dragging the viewport



(a) Swing Viewport Setup **(b)** GraphVisualization Viewport Setup

Figure 8.3: Comparison of the viewport setup provided by Swing (a) and by the GraphVisualization (b). In Swing, the viewport (blue) can only be moved within the *minimal scrollable area* (red). The GraphVisualization adds a margin and allows to place the viewport outside this area. The scrollable area (dotted) is calculated with respect to the viewport position.

with the right mouse button or through Navigator operations. The GraphScrollPane ensures that the client area shrinks automatically when the viewport is moved back into the minimal bounding box.

Each DefaultGraph instantiates a new GraphViewport which handles all coordinate transformations between screen coordinates and graph coordinates. Therefore, the original methods `toScreen` and `fromScreen` of the JGraph class have been overridden. Each GraphViewport exactly works with a single GraphScrollPane. Therefore, the GraphScrollPane takes the DefaultGraph as a parameter of its constructor. Instead of using an observer class which listens to viewport changes, the GraphViewport holds a reference to GraphScrollPane and informs it about state changes. This way it is possible to preserve the order of events and delay updates while zooming as described in the next section.

8.3 Caching Rendering Results

As mentioned in the introduction of this chapter, JGraph applies the flyweight pattern to draw the contents of a graph. The painting of cells starts in the method `paintCells` of the class DefaultGraphUI. The method iterates over all views and configures a *renderer* object. The result of the configuration is then passed to a Swing CellRendererPane as shown in Listing 8.1. In line 6, the renderer is requested from the view which has been passed to this method. Actually, this call configures the renderer with the state flags and the contents of the view. The renderer evaluates these flags and adapts the appearance of the component returned. Afterwards, as shown in line 10, the returned component is passed to the renderer pane.

The algorithm assumes that the returned component is atomic, meaning that it should not consist itself of multiple components like a JList or a JButton. This is because such complex components apply a layout to their children, which is not validated by the algorithm shown in Listing 8.1. In other words, for such components the rendering can produce unpredictable painting errors. Thus, even nodes having complex content must be drawn in a single step.

A typical example for a complex node is the default node representation used in the OntologyVisualization (Figure 7.2(b) on page 47). These nodes show a title and a subset of properties of the resource represented. The problem is that whenever a repaint is required, all properties must be configured again and painted following a certain layout. Smooth animations for zoom and pan are not possible this way.

To solve this problem, the CellViewRenderer interface of JGraph was extended in multiple places. The first extension is the interface ExtendedCellViewRenderer which defines two extra parameters to configure a renderer (Figure 8.2). The first parameter defines the level of detail a renderer should use for its representation. The level of detail is an integer as shown in Listing 8.1. By design contract, higher numbers lead to more detail in the rendering. By default, everything in the Main View is drawn with

```
1  public void paintCell(Graphics2D g2, CellView v, Rectangle2D b,
2      boolean selected, boolean focused, boolean preview,
3      int levelOfDetail, boolean immediateMode)
4  {
5      // configure the renderer
6      Component c = v.getRendererComponent(graph, selected, focused, preview,
7          levelOfDetail, immediateMode);
8
9      // paint cell view (x, y, w and h are queried from the bounding box)
10     rendererPane.paintComponent(g2, component, graph, x, y, w, h, false);
11
12     // paint children
13     if(!v.isLeaf()) {
14         for(CellView c : v.getChildViews()) {
15             paintCell(g2, c, c.getBounds(), selected, focused, preview,
16                 levelOfDetail, immediateMode);
17         }
18     }
19 }
```

Listing 8.1: Shortened version of the paintCell method as implemented in the DefaultGraphUI to demonstrate the flyweight pattern.

level 0 and everything in the navigator is drawn with level -1. Thus, the nodes in the navigator appear as boxes only.

The second parameter is a flag which enables a caching mechanism built on so-called intermediate images. This mechanism is simple but powerful, and supports the rendering of complex nodes including compositing effects while using only a small amount of memory. The idea is to store the result of a drawing in an image along with the current state of the graph. When a repaint request occurs the renderer compares the current state to the stored state and updates the image accordingly. For example, when the background color of a node has changed the image needs an update. Thus the renderer would recreate the image. The same applies when a property was edited or removed. The base classes providing this functionality are CachingNodeViewRenderer and IntermediateImage (Figure 8.2). The class IntermediateImage is a wrapper for an image and the state flags; its main functionality is to provide a simple interface to compare states and create an image if enough memory is available.

However, zooming in this approach turns out to be slower than without caching. This is because the zoom factor of the graph must be stored with the intermediate image to guarantee that the drawing does not become pixelated when zooming in and out. The problem is that zooming usually occurs in small steps and the images would become invalid in each step. Therefore, the update of the images is delayed by a timer. As long as zooming requests occur, the timer is restarted and the renderer returns the outdated version of the image. Thus, this picture is stretched when zooming in and the representation appears pixelated for a short moment. When no further zoom requests occur, the timer triggers a repaint of the graph and the intermediate image is recalculated appropriately.

It should be mentioned that intermediate images require more memory than the original flyweight pattern. Thus, they are only useful for nodes, not for edges which usually span a huge portion of the screen. However, the amount of memory required for an average node is relatively small, even though the images are saved with 32-bit color depth. A node having a size of 130×150 pixels requires about 76kB of memory on the heap. When zooming out, the image size shrinks since its size is scaled by the zoom factor. As a side effect, browsing a huge graph does not become not slower when zooming out.

```
1  public DataObject getLeafDataAtPoint(DefaultGraph g, Point2D p, CellView v){
2      // setup for the given view
3      getRendererComponent(g, v, true, true, false);
4      // initial offset, do not test the title
5      double h = DEFAULT_DIMENSION.height + v.getBounds().getY();
6      // height of a datatype/literal row
7      double sh = getSubObjectHeight();
8      for (int i = 0; i < getSubObjects().size(); i++) {
9          double y1 = h + i * sh;
10         double y2 = h + (i + 1) * sh;
11         if (p.getY() >= y1 && p.getY() <= y2) {
12             return objects.get(i);
13         }
14     }
15     return v.getUserObject();
16 }
```

Listing 8.2: Example implementation of the ComplexNodeViewRenderer interface in Java code. The method returns the data structure at the given point. This is helpful to determine which part of the node should be edited.

8.4 Adapted Inline Editing Support

One of the biggest strengths of the flyweight pattern is the loose coupling between the painting code in the renderer object and the view holding the state information. However, this strength becomes a problem for hit detection when the renderer creates complex results, such as the general node in the OntologyVisualization. The problem results from the fact that a view only stores the bounding box, but makes no assumption about the inner appearance. Usually, this is not required, since a view is assumed to be an atomic element as described in the last section. However, for the QuickEdit tool it is important to know which part of the node is affected by a click, since it must supply different editors depending on that information. In the case of the icon representation the hit test is easily done with the bounding box of the node. In the case of the general node, further tests are required to check whether the title, a literal, or a datatype property is affected.

A solution was required which does not break the loose coupling of view and representation, but supports access to the contents of a node. To achieve this, the interface ComplexNodeViewRenderer was defined (Figure 8.2). This interface defines a single method which must be implemented for the internal hit test. Any node renderer providing a complex representation of its contents should implement this method. Listing 8.2 shows an example for the implementation used in the DefaultNodeRender. The first step is the configuration of the renderer. Afterwards, the algorithm simply loops over a list of all literals and object properties shown for this view. Within this loop the algorithm tests whether the vertical coordinate is contained in one of the "virtual bounding boxes" of these sub-objects. Once the test succeeds, an object is returned with information about the corresponding statement.

8.5 State Pattern for Extensible Tool Support

In JGraph, the class BasicGraphUI is responsible for handling all user input. Internally, this class defines a set of event handlers which react on mouse and keyboard input. This class makes extensive use of internal classes to define the event handling functions. This makes it possible to instantiate a new JGraph with just a few lines of code while getting the full functionality of the package. However, for the GraphVisualization this approach turned out to be too restrictive since it should be used in different contexts. For example, it should be possible to react upon mouse events differently in the StarVisual-

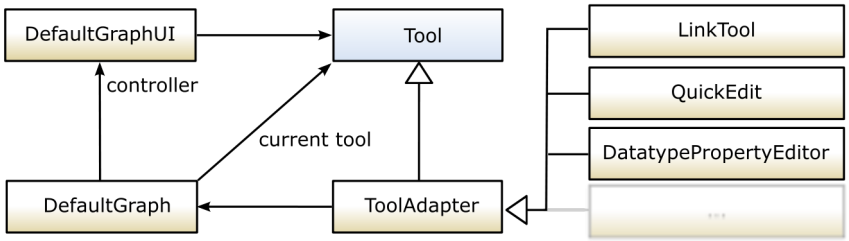


Figure 8.4: Class structure of the tool support in the class DefaultGraphUI. The UI only deals with tools and delegates input events to the currently active tool of the DefaultGraph. The ToolAdapter provides a base class for new tools. Yellow boxes denote classes and blue boxes denote interfaces.

ization and the OntologyVisualization. In the StarVisualization, the node should be expanded, while in the OntologyVisualization the DetailView should be opened or an editing process is started. Support for tools is particularly difficult to implement with these internal classes.

It would have been possible to change the UI for each visualization, but in this case, most functionality would be replicated across the different classes. Thus, to support tools like the LinkTool and editors like QuickEdit, a variant of the well-known state pattern was implemented, as shown in Figure 8.4. First of all, the tools to be used in the visualization must implement the tool interface. This interface extends common Java event handler interfaces like MouseListener, MouseWheelListener and KeyListener. Furthermore, it provides some convenience methods which support the handling of double clicks directly or the activation and deactivation of tools. To keep the tools small, a default implementation of this interface is given by the class ToolAdapter. This class implements the basic activation and deactivation procedures and contains stubs for the methods defined in the Java interfaces.

When activated, a tool registers itself with the DefaultGraph as active. DefaultGraph takes care that only a single tool is activated at a time and deactivates other possibly active tools. If an input event occurs, the DefaultGraphUI queries the active tool from the DefaultGraph and delegates the input event accordingly. If a tool did not consume the delegated event, the DefaultGraphUI performs further processing. For example, it does check whether key or mouse bindings exist which must be informed. Only if there is no event handler, DefaultGraphUI delegates to the JGraph implementation. In essence, the restrictive structure provided by JGraph was turned into a flexible framework.

8.6 Efficient Inspector Implementation

The idea of the Inspector is based on the Excentric Labels of Fekete and Plaisant [1999]. An early implementation was based on the non-crossing lines algorithm described in the paper and is shown in Figure 8.5. This initial implementation was then simplified, since the label connections were found more confusing than helpful to the users. In fact, the connecting lines introduce more clutter to the display than they support orientation. Moreover, the initial implementation did not provide support for simple selection.

The current implementation of the Inspector simplifies the original idea since the labels are directly placed over the nodes. If the Inspector is activated and nothing is selected, the intersected nodes are computed when the mouse is moved. The intersection test uses a simple comparison which is shown in Formula 8.1. Here, r denotes the radius of the current active area, s denotes the current scale factor of the graph, m_x, m_y represents the position of the mouse cursor, and c_x, c_y is the center of the bounding box of a node.

$$\sqrt{(m_x/s - c_x)^2 + (m_y/s - c_y)^2} < r/s \tag{8.1}$$

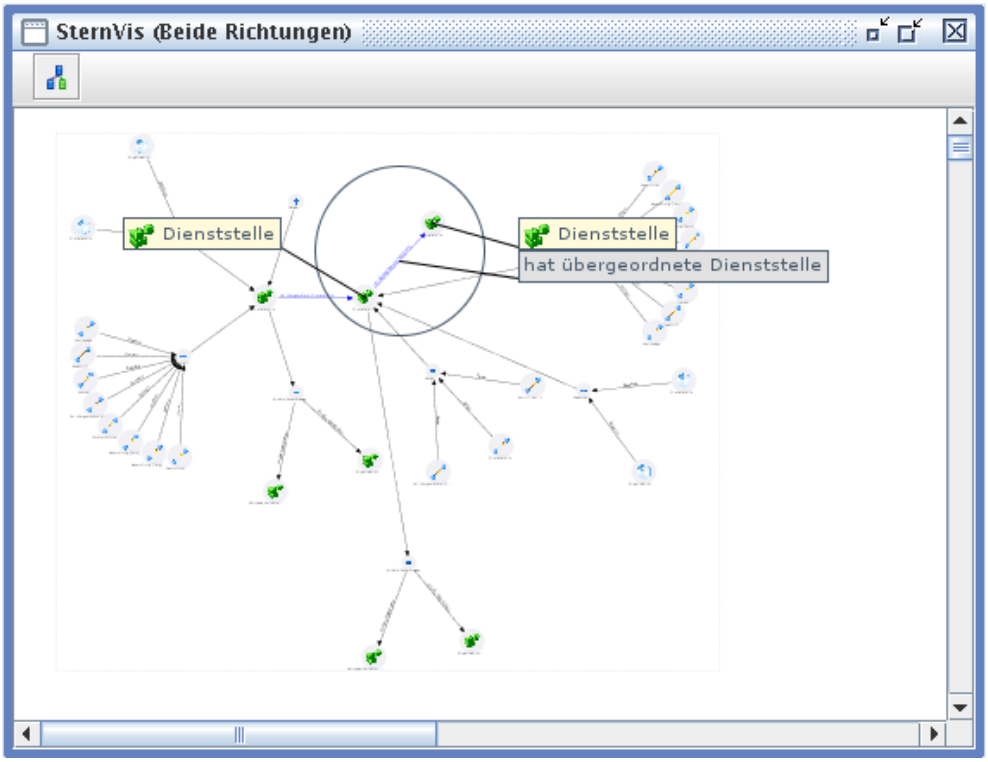


Figure 8.5: An early version of the Inspector based on Excentric Labels (Fekete and Plaisant [1999]). The labels of nodes and edges appear stacked beside the focal area. The approach was scrapped since the connection lines introduce extra clutter.

To sort the intersected nodes, the results are stored in a generic ArrayList of OverlayLabels. An OverlayLabel is a simple wrapper class for CellViews and their distance d to the current mouse position. Since the class implements the java.lang.Comparable interface the list of results can be sorted easily using the Mergesort algorithm of the java.util.Collections class. In essence, the new implementation is fast and more helpful than the original approach.

8.7 Summary

This chapter discussed some of the implementation obstacles which occurred during implementation. When starting the work, JGraph had been considered as a good starting point since many of the requested features were already provided. This assumption needs to be put into perspective.

JGraph is desigened as a component but not as a framework. This subtle difference means, JGraph is easy to use as long as the provided features are sufficient. The adaptation and addition of features often requires more work and is more difficult than it should be. One reason is the abundant use of internal classes. When extending JGraph, new functionality needs to be implemented in internal classes as well to reuse the base functionality. However, this is contradictory to the idea of a framework since internal classes reduce the flexibility.

Another drawback results from the age of JGraph. Its backward compatibility with Java 1.3 prohibits the use of generics which implies a lot of explicit type casts in the code.

Chapter 9

Formative Evaluation

The visualization was evaluated using a thinking aloud test as described by Boren and Ramey [2000]. This chapter is organized in two parts. The first part, Sections 9.1–9.5, describes the test setup including the methodology, participants, tasks, and test environment. The second part describes the results of the test and gives recommendations for improvements of the GraphVisualization Plug-In.

9.1 Test Procedure

Users are given typical tasks while being filmed with a camera. To gain further insight into usability problems and bugs, the users are asked to verbalize their thoughts loudly while they work. The facilitator takes the role of a mostly silent listener, but is allowed to acknowledge the participants thoughts or to fix unforeseen problems. This evaluation method has some characteristics to be aware of when interpreting the results. First, the test setup may be unusual for participants and may influence the way tasks are solved. For example, people are usually not observed when they work. Hence, they could feel uncomfortable with the camera and the mirror being used. Second, the experimenter may influence the result by giving unconscious feedback. As pointed out in Krahmer and Ummelen [2004], this can lead to completing a task even though it is unclear to the user why something worked.

The tests including the interviews and the feedback questionaire were conducted in German, since all participants are native German speakers. A transcription of each test and the original test materials can be found in Appendix B.

9.2 User Groups

As discussed in Chapter 6, the visualization might be used by three different target groups: Developers of m2n, consultants of external companies, and end users. Since most of the developers became familiar with the visualization during its development, the study was designed to test the second target group. These users might benefit from instruction lessons given by m2n, but do not have the insight into framework internals and its concepts.

The participants were given no special training, but were allowed to read the documentation at any time during the test. In the background questionnaire it was ensured that the very basic concepts, like object, class, object property, datatype property and literal, were understood. Table 9.1 lists the persons who participated in the test.

9.3 Tasks

The users were asked to perform the following tasks, whereby each task was assigned a maximum completion time. Each test started with a simple orientation task in the target ontology of a client. At the beginning of each test the database of the client and the server was reset and the system had been restarted. All palettes of the visualization were hidden and the appearance of nodes was set to icon mode. For the first task (Become Acquainted), users were explicitly pointed towards documentation. After 8 to 10 minutes the facilitator posed a few check questions to make sure the user understood the basic concepts of the visualization.

The tasks could be accomplished in a multiple of ways. However, using the tools of the visualization should make any task easier to solve. Since it is quite easy to become lost in the framework configuration, the facilitator sometimes posed questions to redirect the user. However, the need for these questions might indicate that there is room for improvement. The following sections describe the tasks given to the users and the preferred ways to complete them.

9.3.1 Become Acquainted

Task: Make yourself familiar with the visualization. The Wiki may help you!

Preconditions: The visualization is opened and switched to icon mode.

Maximum Time: 8 – 10 minutes

Completion Criteria:

- The user wants to finish the task.

Check Questions:

- What do the nodes mean in the visualization?
- Did you notice the different edges and do you know what they mean?
- Did you notice tools in the visualization?

9.3.2 Find and Describe a Class

Task: Find the class Beitrag. Which datatype properties are related to this class? Which literals are related to this class?

Preconditions: The user knows how to pan and zoom.

Maximum Time: 2 – 4 minutes

Criteria:

- found the class,
- named the datatype properties `gültig von`, `gültig bis`, `ist direkt unter Kategorie`, and `ist qualifizierter Beitrag`,
- named the literals `icon` and `name`.

Possible solutions:

- used the QuickSearch to find the class by its name,
- the tooltips in the Main View,
- used the class tree
- used properties tab of the DetailView

9.3.3 Find and Describe Relations

Task: Find all direct subclasses of Zielontologie. Name the subclasses. If the class Person is connected to one of these classes, name the object properties.

Maximum Time: 3 – 5 minutes

Completion Criteria:

- found the classes Glossareintrag, Informationsobjekt, Dokument, Projekte&Co., Verweis and Wissensträger,
- found the object properties `hat Autor`, `ist Autor von`, `arbeitet an` and `hat beteiligte Person`.

Possible solutions:

- used the QuickSearch to find the class by its name,
- activated the Search Light to highlight related nodes,
- activated the Inspector, the Magnifier, the title scaling, or tooltips for reading the titles.

9.3.4 Create Class

Task: Create a new class Mitarbeiter. Add a datatype property Gehalt using a proper datatype. Add a literal Beschreibung with the content "Festangesteller in einer Organisation".

Maximum Time: 5 minutes

Precondition: The user can distinguish literals, datatype properties, and knows about XML datatypes like float or integer.

Completion Criteria:

- created a new class,
- renamed the class,
- created a datatype property with a numeric type,
- added a new literal.

Possible Solutions:

- created class from the context menu,
- renamed the class using the Quick Edit tool or the DetailView,
- added properties using DatatypeProperty Creator and the LiteralCreator or the DetailView.

9.3.5 Add Object Property

Task: Create a subclass relationship between the classes Mitarbeiter and Person. Create the object property `arbeitet in` between Mitarbeiter and Organisation. Finally, create the object property `hat Mitarbeiter` between Organisation and Mitarbeiter.

Precondition: The user knows that the subclass relation is an object property.

Maximum Time: 5 - 10 minutes

Completion Criteria: Object properties were created and renamed.

Figure 9.1: Test environment at Technologiezentrum Telekom Austria.

Possible Solutions:

- used the LinkTool and the QuickEdit tool,
- used Drag&Drop in the class tree palette,
- used the DetailView and the MultiTree,
- used the link option from the context menu and some other tool to rename the property.

9.4 Test Environment

The test was conducted on a HP notebook running Windows XP SP2. The client and server ran on the same machine and were previously started directly from the Eclipse IDE. Table 9.2 lists the hardware and software used during the setup. The tests took place in the m2n office and a meeting room of a customer. Figure 9.1 shows the test environment as used for TP3, TP4, TP5 and TP6. Users were filmed with a digital camera and a mirror was placed beside the screen to capture the users' facial expressions.

9.5 Interview Questions

In addition to the feedback questionnaire, the following questions were asked (in German) at the end of the test.

1. How was it?
2. Is the visualization comprehensible?
3. Do you have enough possibilities for orientation?
4. Is the usage of the tools intuitive?

9.6 Discussion and Analysis

Users who read the documentation had an easy job to accomplish the tasks even if not familiar with the framework. The same applies for users who had a technical background. The pilot users and TP2 read the documentation carefully. TP3, TP4, TP5 and TP6 did not look at the documentation and needed help during the test. Most of the participants had only little or no experience with the framework and its related technologies (See Table 9.1).

Beside issues found in the visualization other problems occured with the DetailView which should be mentioned first, since they influenced the test. A common problem arose from *inconsistent technical terms* used in the properties tab. For example, users who tried to set the type of a datatype property were often unable to associate the word "range" with a datatype or had problems to see the difference between "range" and "range value type". The second problem is that datatype properties and object properties appear in the same list. Their distinction is only possible using tooltips containing a large amount of text. Another issue is speed when opening the DetailView. For example, TP3 selected six resources at once to show them in the DetailView. This led to a nearly unresponsive system. This impression was supported by the progress indicator which was not animated at this time. To ease this problem, there should be *a warning before long operations* and always an *option to cancel them*.

9.6.1 Orientation Tasks

Tasks 1–3 can be seen as typical orientation tasks which can easily be solved using the QuickSearch and the Inspector. Alternatively, the palette showing the class tree may be used to help browse the contents of the ontology.

As shown in the transcripts in Appendix B.5, all participants used the QuickSearch or tooltips. The automatic centering of the search results when selecting them from the result list was appreciated very much. A problem resulted from the colors. TP4 mixed up the green color used to mark search results with the green color of the selection border. Hence, there should be a way to *let the user choose the marking color*. TP4 tried to use the Quick Search to look for multiple classes. At the moment this is only possible when using regular expressions. Even so, the QuickSearch does not give any explanation about the syntax of these expressions. A *tooltip for the search term* input field could ease this problem.

Some users did not understand or did not see the buttons at the bottom and requested to make them more visible. One problem arose from the tooltips which became visible immediately over a state button. As long as the tooltip was visible, the state could not be activated because it interfered with the mouse event. This problem seems to be caused by the Swing ToolTipManager which becomes active immediately after the mouse leaves the Main View. Therefore, the visualization might have to *install its own tooltip manager*.

9.6.2 Modeling Tasks

Task 4 and Task 5 are typical modeling tasks which can be carried out using the DetailView, the Multi-Tree, or the tools provided in the visualization. All users who read the documentation made use of the tools. All others tried to accomplish the task the old way at first.

Participants TP3 to TP6 did not look at the documentation and tried to finish their job using the DetailView. In most cases they got stuck with this approach and the facilitator asked whether they see an easier way to continue. Except for TP3 all users found the tools in the visualization but were often confused how to start editing or adding properties. It was unclear that a double click is needed to start editing. Most users selected the resource and clicked the tool afterwards. If nothing happend, they checked the context menu for some method to start editing. TP2 suggested that it would be easier to

know that a tool is active by *changing the cursor*. This improvement can easily be integrated since the Tool interface discussed in Chapter 8 already supports this feature.

The context menu contains several menu items depending on the users' rights and the selected resource. One of the actions is "Neu" which creates a new object. Users who did not read the documentation mostly confused this point with the submenu "Erstellen...". The problem results from the fact that a new resource is only visible in the target ontology if it is a subclass of the target ontology. No participant was aware of this problem in advance and the creation mostly failed the first time.

Most users understood immediately how the LinkTool can be used, but did not realize that a subclass relation is an object property as well. The problem resulted from the fact that the tooltip of the LinkTool did not mention "subclass" but only object properties. Thus, a simple *change to the tooltip* may help to overcome this pitfall. TP5 tried to drag with the mouse during the link process due to experience from other modeling tools. Thus, the statebar may be changed to contain more useful hints for the modeling process such as "select subject of the statement" and "select object of the statement".

9.7 Interviews

After the test, an interview was conducted using the questions in Section 9.5. One of the first impressions was that the visualization is slow. However, in fact it was not the visualization which was slow, but the client when users opened the DetailView or MultiTree multiple times. Another impression was that the visualization is comprehensible and intuitive to handle once all functions were found. Two users remarked that the tasks were unclear from the terminology being used and that it was therefore not possible to finish a task.

When asked about the quality of the visualization, most users appreciated the presentation as sufficient and nice. However, when asked in more detail most participants could not distinguish between datatype properties and literals. Only users familiar with logic or modeling background had no problems with the distinction. Maybe this issue could be solved by a simple *change in the tooltip* or a *symbol in the renderer*.

Another issue concerns the size of the nodes. Some users wanted to change the size of the node manually in order to read the title or the properties in full. Admittedly, this is quite natural but is not that easy to integrate currently. A first improvement could be made by *changing the default width to a larger value*.

Very positive feedback was given about the possibilities of orientation. Most users noticed the Navigator and the QuickSearch. Participants also rated the Search Light and the Inspector as very helpful. Only one user found the presentation in the Navigator not sufficient due to the lack of details. The user was familiar with CAD programs and was looking for a possibility to zoom into a certain area by selecting the region. In essence, a *zoom tool which lets the user select a rectangular region to be zoomed was requested*. Such tool could be integrated into the next version of the GVP.

Most participants did not understand the distinction between tools and states at a glance. The high scores in Table 9.3 should be regarded with respect to the time users got after the interview. Participants who did not read the manual and had no background in modeling or logic had problems to understand the tools. A first complaint concerned the visibility of certain functionality. The icons were observed as too small or not meaningful enough. It was proposed to *add further descriptions to the toolbar* or the icon itself. Moreover, *any function given in the toolbar should also be available in the context menu*. The LinkTool was regarded as very easy to use, but the activation of other tools was experienced as unusual. Most users expected to select a node and start the activation by clicking the button in the toolbar. As the cursor of the mouse did not change, they waited for some kind of input field to appear. This confusion might result from the fact, that tools and palettes are hosted in the same container. In summary, once it was understood that tools start with a double-click, their handling was found practical.

9.8 Feedback Questionnaires

Table 9.3 shows the results of the feedback questionaires conducted at the end of the test after finishing the interview. The original questions can be found in Appendix B.4. The neutral scale given in the original materials was converted such that 6 maps to the most positive score and 0 maps to the most negative score. If no answer was given, the table cell contains "–". The last two columns show the average score for a question and the standard deviation.

All participants were allowed to use the GraphVisualization after the test was finished in order to ensure they used every tool. Therefore, the scores in Table 9.3 should be regarded with respect to the answers given in the interview.

9.9 Positive Findings

1. Good Orientation Support

As shown in the transcripts (Section B.5), all participants made use of the QuickSearch. The automatic placement of the viewport to center the selected result was rated very helpful. The second most often mentioned point was the Search Light and the Inspector. Activating the search light often resulted in an aha experience by the user because they saw immediately which resources are related.

2. Tools Simplify Modeling

Even though not all participants were aware of the available tools during the tasks, most found they could support their modeling process when trying them during the interview. Often they found out how to use a tool during the interview. Participants who had experience with older versions of the framework and knew the old-fashioned way of modeling were excited about the simplicity. TP5 especially mentioned the simplicity compared to products like IBM Rational when creating classes and object properties.

3. Appealing and Informative Node Presentation

Most users found the UML styled nodes informative and visually appealing. TP5 noted the use of colors for selection and the alternating colors for datatype properties and literals. Non-technical users could distinguish between the different kind of edges used to represent different object properties.

9.10 Recommendations

Improve Tool Access and Visibility

User (Timestamp): P2 (00:27:10), TP2 (00:47:10), TP4 (01:04:10), TP6 (01:05:00)

As described above, most users tried to start the editing process of a resource using the context menu or had problems to start the editing through double clicking. TP2 suggested *changing the cursor* to make the user aware of different mouse functionality. TP4 suggested *adding a context menu group* reflecting the state of the tools exactly. Any tool in the toolbar should also be available in the context menu. This should also shorten the mouse movements and increase the speed of operation in the visualization.

P2 and TP6 proposed *adding descriptions* to the tool buttons to be aware of them without reading the manual. TP6 gave further hints how the icons should become more visible. He suggested *improving the icons* of the buttons by some sort of state lights to make the user aware of the currently active tool. P2

suggested that *editing should start immediately* after creating a resource. However, it is unclear where the editor should initially be placed. The center of the edge is not sufficient as it may not be in the viewport. Moreover, it predetermines the workflow of the user. Therefore, this recommendation might be a point of discussion.

Improvement for Representation

User (Timestamp): P1 (00:38:30),TP6 (00:58:40, 01:03:00)

A common behavior of users to gather information about properties was to open the DetailView and read the properties from the properties tab. Only TP5 was aware of the tooltips provided in the visualization and used them properly. A simple improvement may result from a *common icon preceding the literals and datatype properties* in general node presentation mode.

P1 and TP6 explicitly mentioned that the *nodes should be resizeable*. They tried to resize the nodes to read the terminal slots. However, in this case, useful boundaries for minimum and maximum width must be found.

TP6 mentioned that the possibility to mark colors visually is important, but suggests *further colors for better distinction of object properties*. This recommendation must be implemented with care. The good visibility of associated resources results from the contrast of the yellow color to the rest of the graph. Introducing more colors can lead to clutter and reduce the effectiveness of the tool as explained in Chapter 4.

9.11 Summary

This chapter discussed the procedure and the results of the thinking aloud test conducted for the evaluation of the m2n IMF GraphVisualization Plug-In. Although non of the participants had experience in usability testing, none of them felt uncomfortable with the test setup.

The test not only revealed usability problems of the visualization but of the whole client like false or missing localization, and poor feedback when triggering operations on many resources at once.

Regarding to the m2n IMF GraphVisualization Plug-In the feedback given is very positive. The biggest improvements result from the easiest additions like the QuickSearch, the Search Light and the Inspector. The Navigator improves the workflow as the users are familiar with the concept from other software. The editing tools become usable due to their simple structure which lets the user intuitively understand how to use them.

Recommendations concerned missing tools, missing possibilities to access the tools from the context menu, and the ability to adapt the user interface. Especially the icons and the description used could be improved to make functionality understandable without reading the manual.

The test makes clear that the client needs a user interface guideline to improve the consistency of names, labels and icons. Currently this part is left to the developers of m2n.

	P1	P2	TP1	TP2	TP3	TP4	TP5	TP6
General Information								
Date of Test in 2008	07 Oct	08 Oct	11 Oct	12 Oct	15 Oct	15 Oct	15 Oct	15 Oct
Time	17:00	10:10	14:35	10:11	10:48	12:40	14:28	15:38
First Name	Thomas	Stefan	Michael	Jörg	Harald	Robert	Markus	Ronald
Gender	male	male	male	male	male	male	male	male
Age	32	26	27	27	37	38	25	29
Education	Degree, Trading	Student, Informatics	Degree, Informatics	Degree, Informatics	A-levels	Degree, Physics	Student, Business Informatics	A-levels
Background Knowledge								
Experience with Computers (years)	15	10	15	8	14	15	10	8
Occupation	Management	QA	Development	Research	Management	Analyst	IT Support	Consultant
Object Orientation	○	●	●	●	○	●	●	●
Class	●	●	●	●	○	●	●	●
Semantic Graph	●	●	●	●	●	○	○	○
Ontology	●	●	●	●	●	○	○	○
RDF	●	●	○	●	●	○	○	○
OWL	●	●	○	●	●	○	●	○
Object Property	●	●	○	○	○	○	●	○
Datatype Property	●	●	○	○	○	○	●	○
Literal	○	●	○	○	○	○	●	○
Special Knowledge								
Knows Modeling Tools	no	Protégé	no	no	no	no	Rational Rose	no
Previously used m2n Framework	Testing	Testing	no	no	Become Acquainted	no	no	Become Acquainted
Previously used Visualization for	Orientation	Orientation	no	no	no	no	no	Orientation
Usability Test Experience								
Participated	no	no	no	no	no	no	no	no
Test Team Member	no	no	no	no	no	no	no	no

Table 9.1: Overview of the persons who took part in the thinking aloud test. Two pilot tests (users P1 and P2) were conducted, followed by the real test with six test users (TP1 to TP6). Symbols: ● = knows the concept, ○ = does not know the concept.

Hardware

Computer	HP Compaq nx7400 Notebook, 3 GB RAM
Monitor	iiyama ProLite E2003WS
Monitor Resolution	1.680 x 1.050
Monitor Size	22" TFT
Colors Depth	24Bit
Camera	Panasonic NV-GS75 (miniDV)

Software

Operation System	Windows XP Professional, SP2
Java, JRE	Java 1.6.0_04, JRE build 1.6.0_04-b12
Eclipse	Eclipse 3.3.2 (Europa)
m2n	Trunk, Rev. 15018

Location and Dates

Pilot 1	Meldemannstraße 18 / 1210 Vienna, 08 Oct 2008
Pilot 2	Meldemannstraße 18 / 1210 Vienna, 09 Oct 2008
TP1	Meldemannstraße 18 / 1210 Vienna, 11 Oct 2008
TP2	Meldemannstraße 18 / 1210 Vienna, 12 Oct 2008
TP3–TP6	Technologiezentrum Telekom Austria, 1030 Vienna, 15.10.2008

Table 9.2: Test equipment and setup used for the thinking aloud tests.

	P1	P2	TP1	TP2	TP3	TP4	TP5	TP6	μ	σ
1. Information content of nodes is good	4	3	5	5	6	3	6	6	4.75	1,64
2. Visualization is fast enough	4	6	1	6	5	5	6	6	4.86	2,98
3. Edge appearance is understandable	5	5	4	6	4	3	6	4	4.43	1,13
4. Object property creation is intuitive	4	6	4	4	1	6	4	6	4.38	2,84
5. Datatype property creation is intuitive	4	6	0	2	1	5	6	5	3.63	5,41
6. Editing is intuitive and simple	5	4	4	3	3	5	5	5	4.25	0,79
7. Tools have reasonable names	5	4	6	5	6	4	6	2	4.75	1,93
8. Tools are arranged logically	1	5	5	6	6	4	6	4	4.63	2,84
9. Tools could help modeling	4	6	4	6	6	5	6	6	5.38	0,84
10. Combination of tools helpful	3	6	5	–	3	–	6	6	4.83	2,17

Table 9.3: Results of the questionnaires conducted after each test. The full questions can be found in Appendix B.4. The answers were mapped from the original scale to a score between 0 and 6. Six points mean absolute agreement. Zero points mean absolute disagreement. The last two columns show the average value (μ) and the standard deviation (σ).

Chapter 10

Outlook

In the course of this work the m2n GraphVisualization Plug-In (GVP) was adapted multiple times for the needs of different customers. The abilities in configuration and presentation grew constantly such that the GraphVisualization can be configured for different contents and available functions. Even though the navigation features work in every visualization, the best support for editing is currently provided for graphs which are similiar to the OntologyVisualization. Thus, there is still much room for improvement.

First, the creation of object properties could be further improved. For example, it is not possible to create object properties having a certain cardinality using the LinkTool. It is still required to open the DetailView to enter the values. This use case was not covered during requirements collection, since object properties without cardinality are found to be sufficient in most cases. In other cases the cardinality of these properties is likely to be restricted by the schema itself. Another improvement could be a tool which supports the marking of symmetric properties. In both cases the conceptual model developed in Chapter 6 supports the addition of these extensions very easily in later revisions. The cardinality can be added as part of the QuickEdit tool. The editor component would need to be enhanced by further input fields. To mark properties as symmetric, a new tool could be added. Such symmetric properties must also be visually marked using color coding or glyphs.

A second group of improvements concerns the modeling of rules which were not discussed in detail in this thesis. Tools like the LinkTool and QuickEdit also provide a major improvement for the modeling process in RuleVisualization. The same applies for the rule block palette which serves as a container for producible rule elements which can be dropped into the visualization. However, these improvements uncovered other pitfalls in the typical workflow of a developer. For instance, it is not easily possible to assign a value to variables and constants. Furthermore, most methods are still missing a description, which makes it impossible for an external consultant to model a rule.

The third field of improvement concerns the presentation of the graph. Significant effort was put into providing a clean and comprehensible user interface and graph node representation. However, edge presentation did not receive as much as attention. Hence, edge labels lack of readability when zooming out. Only the Inspector provides some help. Experiments were made with semantic fisheye views which can be applied to nodes and edges. With respect to the currently selected node, a level of detail was applied to associated resources. However, it is still unclear how to choose the correct level of detail for a representation. There is no value in showing a node as a semi-transparent box, if the speed is not improved and the user is forced to change the focus continuously when modeling graphs. Instead, the visualization should provide better support for continuous refinement. The user should be able to select a part of the graph and hide other parts.

Lastly, there is also room to simplify the configuration of the component. Currently, the major way is to write the N3 files or to open the configuration resources and manipulate them in the DetailView and the MultiTree. However, this is a problem which must be solved for all m2n components, since each of them introduces its own schema which requires a deep understanding of the implementation.

Chapter 11

Concluding Remarks

This thesis presents the ongoing development of the m2n GraphVisualization Plug-In (GVP) which is now a framework for the visualization and manipulation of semantic graphs in different applications. The overall motivation for this thesis was the improvement of the existing visualization Plug-In in terms of visual representation and modeling support. Therefore, Chapter 2 gave an introduction to semantic graphs and discussed their foundations in logic, philosophy, and cognitive science. Chapter 3 introduced the m2n Intelligence Management Framework and explained its very basic concepts like the GUI Plug-In and previously available visualization methods. Chapters 4 and 5 gave a review of common information visualization techniques and recent interactive and manipulation tools. Requirements were collected based on the existing visualization, informal interviews, and interim development. The findings from this analysis were compiled into a user model and an extensible conceptual model described in Chapter 6.

Chapter 7 presented the redesigned component. It was shown that a powerful, but also simple, user interface is possible to visualize semantic graphs in diffent applications. Knowlegde from information visualization was used to improve the presentation of the graph. The new GraphVisualization Plug-In supports features like linking & brushing, fast navigation, search, as well as detailed inspection at low zoom levels. Some specific details of the implementation are described in Chapter 8.

The implementation was evaluated in a thinking aloud test with eight users. The results and the methodology are documented in Chapter 9. Most of the participants provided constructive feedback. The results convey that the effort is going in the right direction. Even if there is still much to do, the fundermentals for a usable GVP are laid. Chapter 9 and 10 described some potential points for further enhancements.

Appendix A

Standard Namespace Definitions

This appendix lists the namespaces for the N3 listings shown in this thesis. The definitions are given in alphabetical order. Each namespace given below must be added to the respective listings using the `@prefix` directive. Note that most defintions follow the conventions for URIs but do *not* point to an internet location as a URL would.

block m2n://method/at.m2n.IntelligenceManagement.guiPlugin.client.block.

button m2n://property/at.m2n.IntelligenceManagement.guiPlugin.visualComponent.button#

component m2n://class/at.m2n.IntelligenceManagement.guiPlugin.visualComponent.

config http://m2n.at/graphVis/config/

edge http://m2n.at/graphVis/edge/

ex http://www.example.org/

foo http://www.foo.com/

graph http://m2n.at/graphVis/

gv m2n://property/at.m2n.IntelligenceManagement.graphVis.component.GraphVisualization#

gvc m2n://property/at.m2n.IntelligenceManagement.graphVis.component.

ifparam http://m2n.at/2006/03/rule-schema/block/If/parameter/

mapping http://m2n.at/2006/03/rule-schema/mapping/ .

mappingClass http://m2n.at/2006/03/rule-schema/mapping/Class/ .

node http://m2n.at/graphVis/node/

outparam m2n://method/at.m2n.IntelligenceManagement.guiPlugin.client.block.MethodGetSelectedListItem/parameter/

owl http://www.w3.org/2002/07/owl#

owlx http://m2n.at/2005/05/owlx-syntax-ns/

pos http://m2n.at/graphVis/config/pos

rdf http://www.w3.org/1999/02/22-rdf-syntax-ns#

rdfs http://www.w3.org/2000/01/rdf-schema#

rule http://m2n.at/2006/03/rule-schema/

vc m2n://property/at.m2n.IntelligenceManagement.guiPlugin.visualComponent#

window m2n://property/at.m2n.IntelligenceManagement.guiPlugin.visualComponent.window

xsd http://www.w3.org/2001/XMLSchema#

Appendix B

Original Test Materials

This appendix includes the original materials used in the thinking aloud test of the GraphVisualization Plug-In (GVP). The tests were conducted in German, since all users speak German as their first language.

B.1 Vertraulichkeits- und Einverständniserklärung

Danke, dass Sie an meiner Studie teilnehmen. Bitte beachten Sie, dass Ihnen unter Umständen vertrauliche Informationen zuteil werden und dass Sie diese **nicht** weitergeben dürfen. Von der Sitzung werden Bild- und Tonaufnahmen gemacht, um es anderen, die heute nicht anwesend sein können, zu ermöglichen, aus Ihrem Feedback Nutzen zu ziehen. Bitte lesen Sie die folgende Einverständnis-Erklärung und unterschreiben Sie an der dafür vorgesehenen Stelle. Vielen Dank!

Erklärung

Ich erkläre, keine Informationen aus der Studie weiterzugeben. Ich weiß, dass Bild- und Tonaufnahmen von meiner Sitzung gemacht werden. Ich gebe die Erlaubnis, diese Aufnahmen für Lehrzwecke und im Rahmen wissenschaftlicher Forschung zu verwenden.

Ort: _____

Datum: _____

Name: _____

Geburtsdatum: _____

Unterschrift: _____

B.2 Hintergrundbefragung

Danke dass Sie sich als Freiwilliger für meinen Test zur Verfügung stellen. Bitte beantworten Sie die folgenden Fragen:

1. Angaben zur Person

   Geschlecht: □ männlich □ weiblich

   Alter: _____

   Ausbildung:

   □ Lehre
   □ Matura
   □ Studium (Fachrichtung: _____)
   □ Doktorat

   Sehvermögen:
   Verwenden Sie eine Sehhilfe bei der Arbeit am Computer?

   □ Keine
   □ Brille
   □ Kontaktlinsen
   □ Sonstige _____

   Sind Sie farbenblind?
   □Nein □ Ja, und zwar

2. Umgang mit Computern

   (a) Wie lange benutzen Sie bereits Personal Computer?
   _____ Jahre

   (b) Wie würden Sie Ihren Aufgabenbereich beschreiben (z.B. Entwickler)?
   _____

3. Hintergrundwissen

   (a) Markieren Sie bitte alle Begriffe, die Sie erklären könnten:

   □ Ontologie
   □ OWL
   □ RDF
   □ Object Property
   □ Datatype Property
   □ Literal

   (b) Haben Sie Erfahrung im Umgang mit Modellierungswerkzeugen für semantische Graphen?

   □ Ja, und zwar mit _____
   □ Nein

   (c) Haben Sie bereits mit dem m2n Framework gearbeitet?

   □ Ja
   □ Nein

(d) Haben Sie bereits mit der GraphVisualization gearbeitet?

- □ Ja
- □ Nein

(e) Wenn Sie bereits mit der GraphVisualization gearbeitet haben, wozu haben Sie sie verwendet?

- □ Modellierung
- □ Orientierung
- □ Sonstiges: _____

4. Haben Sie schon einmal an einer Usability Studie teilgenommen?

- □ Ja, als Testperson _____ mal.
- □ Ja, als Mitglied im Testteam _____ mal.
- □ Nein.

B.3 Orientation Script and Tasks

Aufgabe 1 – Vertraut machen

In der ersten Aufgabe geht es darum sich mit der Visualisierung vertraut zu machen. Dazu habe ich Ihnen bereits den m2n Client mit der Visualisierung der Zielontologie gestartet. Zusätzlich ist der Browser mit dem m2n Wiki geöffnet, so dass Sie jederzeit die Dokumentation lesen können. Ich werde Ihnen nach ca. 8 Minuten einige Fragen stellen um sicher zu gehen, dass der Rest des Tests funktionieren kann. Wenn Sie fertig mit der Aufgabe sind, können Sie auch vorher Bescheid geben.

► Machen Sie sich mit der Visualisierung vertraut. Das Wiki kann ihnen dabei behilflich sein.

Aufgabe 2 - Finden und Beschreiben von Klassen

In der zweiten Aufgabe soll einfach eine Klasse beschrieben werden.

► Finden Sie die Klasse Beitrag. Welche Datatype Properties besitzt die Klasse? Welche Literale besitzt die Klasse?

Aufgabe 3 - Finden und Beschreiben von Beziehungen

In der dritten Aufgabe sollen Beziehungen zwischen Klassen analysiert werden.

► Finden Sie alle direkten Unterklassen der Klasse Zielontologie. Bitte nennen Sie die Unterklassen! Wenn die Klasse Person mit einer dieser Unterklassen verknüpft ist, nennen Sie bitte die entsprechenden Object Properties

Aufgabe 4 - Eine Klasse Erstellen

Die nächste Aufgabe besteht darin, eine Klasse zu erstellen und einige Eigenschaften anzulegen.

► Erzeugen Sie eine neue instanziierbare Klasse Mitarbeiter. Fügen Sie ein Datatype Property Gehalt mit einem geeigneten Datentyp ein! Fügen Sie ein Literal Beschreibung mit dem “Inhalt Festangesteller in einer Organisation” ein!

Aufgabe 5 - Object Property Anlegen

In der Letzen Aufgabe geht es darum die angelegte Klasse zu verknüpfen.

► Erzeugen Sie eine Unterklassenbeziehung zwischen Mitarbeiter und Person. Erstellen Sie das Object Property `arbeitet in` zwischen Mitarbeiter und der Klasse Organisation. Legen Sie auch das inverse Property `hat Mitarbeiter` an.

B.4 Feedback Formular

Bitte bewerten Sie die Visualisierung mit Hilfe der folgenden Fragen. Der Wert 0 steht für eine neutrale Bewertung. Der Wert 3 steht für volle Zustimmung.

1. Der Informationsgehalt der Knoten ist sehr gut
   Trifft zu | 3 | 2 | 1 | 0 | 1 | 2 | 3 | Unbrauchbar, es fehlen Informationen
2. Die Geschwindigkeit der Visualisierung ist ausreichend schnell
   Trifft zu | 3 | 2 | 1 | 0 | 1 | 2 | 3 | Nein, zu langsam
3. Die Darstellung der Kanten ist leicht verständlich
   Trifft zu | 3 | 2 | 1 | 0 | 1 | 2 | 3 | Unbrauchbar
4. Das Anlegen der Object Properties ist intuitiv und einfach möglich
   Trifft zu | 3 | 2 | 1 | 0 | 1 | 2 | 3 | Unverständlich
5. Das Anlegen von Datatype Properties ist intuitiv möglich
   Trifft zu | 3 | 2 | 1 | 0 | 1 | 2 | 3 | Nein, absolut unverständlich
6. Das Editieren ist intuitiv möglich
   Trifft zu | 3 | 2 | 1 | 0 | 1 | 2 | 3 | Nein, absolut unverständlich
7. Die Werkzeuge sind verständlich bezeichnet
   Trifft zu | 3 | 2 | 1 | 0 | 1 | 2 | 3 | Nein, ich konnte die Werkzeuge nicht zuordnen
8. Die Werkzeuge sind logisch angeordnet
   Trifft zu | 3 | 2 | 1 | 0 | 1 | 2 | 3 | Nein, ich würde es absolut anders machen
9. Die Werkzeuge könnten mir helfen bei der Modellierung
   Trifft zu | 3 | 2 | 1 | 0 | 1 | 2 | 3 | Nein, mir fehlen die wichtigsten Funktionen
10. Die Kombination der Werkzeuge (Baum, Visualisierung) ist hilfreich
    Trifft zu | 3 | 2 | 1 | 0 | 1 | 2 | 3 | Nein, verwirrend

Positive Eindrücke:

- _____
- _____
- _____

Verbesserungsvorschläge:

- _____
- _____
- _____

B.5 Test Transcripts

This section contains the transcripts of the thinking aloud tests in chronological order. The first two tests (users P1 and P2) were conducted as pilot tests to see if the questions and the setup work. Since the questions were not fully comprehensible the first time, they were reformulated and tested with the second pilot user. Thus, the transcript of the first test differs slightly from the other transcripts. The following symbols are used:

► A new task begins.

☺ The facilitator speeks or interrupts.

ϟ A bug occurred in the program.

B.5.1 P1 (Thomas)

	Time	Event
	(00:00:00)	Introduction thinking aloud test
	(00:02:06)	Background questionnaire
►	(00:05:41)	**Task 1**
☺	(00:05:43)	First question
	(00:06:40)	Reads documentation about main window
	(00:07:20)	Finds how to zoom
	(00:07:38)	Uses scrollbars to pan
	(00:07:57)	Complains about slow mouse
	(00:09:30)	Moves nodes, to remove a subclass edge (does not work)
	(00:10:00)	Seems to be confused about toolbar and Statebar
	(00:11:20)	Finds a way to switch from icon mode
	(00:12:45)	Question: where are the properties?
	(00:13:30)	Creates an abstract subclass
	(00:14:00)	Tries to find out how to create a new object property
	(00:14:37)	Confused about documentation for datatype properties
	(00:16:22)	Answers control questions
►	(00:18:23)	**Task 2**
	(00:19:19)	Finds class using tooltips
	(00:20:00)	Requests explanation about literals
	(00:20:20)	Names literals correctly using tooltips
►	(00:20:30)	**Task 3**
	(00:21:20)	Moves to Zielontologie node with simple pan
	(00:21:30)	Reads documentation to find tool that could help to show dependencies
	(00:22:04)	Finds Inspector in the documentation
	(00:22.22)	Zoomes out and notices the resources that are connected
	(00:22:50)	Names direct subclasses using the tooltip
	(00:24:00)	Wonders where the class Person is
	(00:24:19)	Does not understand the question
☺	(00:24:20)	Cancels question
►	(00:24:38)	**Task 4**
	(00:25:10)	Wonders how to create subclass and reads the tooltips in the Statebar
	(00:25:20)	Looks into the documentation to look for a tool that can be used to create the class
☺	(00:26:12)	Gives hint the context menu
	(00:26:40)	Creates a new subclass using the context menu

	(00:27:10)	Looks into the context menu to rename the class
	(00:27:20)	Reads the documentation
	(00:28:00)	Starts editing the name using the QuickEdit tool
	(00:28:18)	Uncertain about property name and property description
	(00:29:04)	Looks into context menu for possibility to add datatype property
	(00:29:24)	Finds DatatypeProperty Creator and starts editing the resource, mentions double click
	(00:30:14)	Uncertain about the task (adding description property)
☺	(00:30:23)	Explains task again
	(00:30:32)	Uses LiteralCreator to add the literal
ϟ	(00:31:00)	Adding literal removes name property
	(00:31:28)	Wants to add resize the node
	(00:31:40)	Sets the name again
►	(00:32:05)	**Task 5**
	(00:33:10)	Starts creating a datatype property
	(00:33:20)	Notices mistake
	(00:33:50)	Looks for a way to create an object property
	(00:34:07)	Finds LinkTool
	(00:34:35)	Uncertain about the direction of the subclass property
	(00:34:50)	Creates subclass relationship
	(00:35:10)	Wants to create object property using the LinkTool
	(00:35:50)	Sets the wrong subject node, wants to cancel
	(00:36:00)	Wants to rename, double click opens the DetailView by accident
	(00:36:45)	Wonders why object property is straight edge
	(00:37:05)	Interview
	(00:40:00)	Feedback questionnaire

B.5.2 P2 (Stefan)

	Time	Event
	(00:00:00)	Introduction
	(00:00:25)	Consent form
	(00:01:25)	Background questionnaire
	(00:06:20)	Demo thinking aloud test
►	(00:07:50)	**Task 1**
	(00:08:30)	Clicks topic about Search Light tool
	(00:08:47)	First task end
☺	(00:08:51)	Control questions
►	(00:09:50)	**Task 2**
	(00:10:00)	Opens the QuickSearch and inputs the name
	(00:10:10)	Double clicks the result, comment about animation
	(00:10:21)	Opens the DetailView to look for Datatype Properties
	(00:10:24)	Closes DetailView before loading completely
	(00:10:33)	Opens the QuickInfo palette to look for properties
	(00:11:04)	Opens DetailView again and flips the tabs
	(00:11:50)	Observes the datatype properties and literals
	(00:12:00)	Not sure about difference of datatype properties and literals
►	(00:13:05)	**Task 3**
	(00:13:16)	Uses QuickSearch again
	(00:13:32)	Clicks subclass edges

	(00:13:42)	Activates Search Light
	(00:13:42)	Tries to use context menu to identify subclasses
	(00:13:42)	Identifies classes but clicks them - does not wait for tooltip
	(00:14:44)	Confused about Search Light
	(00:15:50)	Opens another visualization
	(00:16:20)	Confused about the information palette
	(00:16:40)	Uses tooltips to name the classes
	(00:17:26)	Uses QuickSearch to search for Person
	(00:18:20)	Identifies the related class
►	(00:19:06)	**Task 4**
	(00:19:20)	Creates new subclass via context menu
	(00:20:00)	Opens the Navigator
	(00:20:10)	Looks for tools in the Statebar
	(00:20:33)	Edits name
	(00:20:50)	Uncertain how to start adding a datatype property editing
	(00:21:03)	Understands how to start editing
►	(00:22:07)	**Task 5**
	(00:22:39)	Activates LinkTool and selects start node
	(00:22:51)	Uses QuickSearch
	(00:23:05)	Uncertain about direction of subclass edge
	(00:25:00)	Wants to see start the editing process directly
	(00:25:10)	Interview
	(00:28:30)	Feedback

B.5.3 TP1 (Michael)

	Time	Event
	(00:00:00)	Consent form
	(00:00:56)	Introduction
	(00:02:29)	Background questionnaire
	(00:06:23)	Demo thinking aloud test
►	(00:07:40)	**Task 1**
	(00:08:20)	Scrolls and notices zoom
	(00:08:36)	Wants to remove selection by clicking an empty area
	(00:08:42)	Notices how to select
	(00:08:55)	Notices how that zoom follows the mouse cursor
	(00:09:10)	Observes that edges are directly selectable
	(00:09:26)	Opens DetailView wonders how to close
	(00:09:50)	Finds the Navigator as overview but does not interact
	(00:10:30)	Looks into the legend of the icons
	(00:11:20)	Finds node description in the Wiki
	(00:11:38)	Finds class tree
	(00:11:41)	Wants to finish the task
►	(00:11:50)	**Task 2**
	(00:12:00)	Uses the QuickSearch
	(00:12:05)	Double clicks the result list, thinks that double clicking nodes centers the result
	(00:12:24)	Tries to use the DetailView to answer question about datatype properties
☺	(00:13:30)	Control questions
	(00:14:00)	Observes that subclass edge has no label

	(00:14:38)	Notices that he does not know how to find literals
☺	(00:14:52)	Shows how to switch to general mode
	(00:15:20)	Notes that the icon switching had been seen in the Wiki
►	(00:15:24)	**Task 3**
	(00:15:50)	Uses the QuickSearch to search the target ontology root node
	(00:16:05)	Irritated due to overlapping nodes
	(00:16:18)	Switches to icon mode, notes that labels are hardly readable
	(00:16:54)	Notes that nodes can be moved
	(00:17:15)	Uses Navigator to move the viewport
	(00:18:04)	Looks for a better possibility to find related classes
ϟ	(00:18:10)	Notices inconsistency between Wiki and tooltip
	(00:19:00)	Activates Search Light and Inspector to name classes
	(00:19:30)	Uses QuickSearch to find class Person
☺	(00:20:00)	Interrupts task
►	(00:20:10)	**Task 4**
	(00:20:30)	Uses new action from the context menu to create a new resource
	(00:20:50)	DetailView opens twice since the system was too slow
	(00:21:30)	Creates a new Datatype Property using the DetailView
	(00:22:00)	Tries to find the correct tab to enter the data
	(00:23:16)	Looks into the Wiki
	(00:24:00)	Notices that the system is slow
►	(00:26:00)	**Task 5**
☺	(00:24:00)	Task change to check the LinkTool
	(00:26:30)	Opens the DetailView and tries to add the relationship
	(00:37:00)	Reads the Wiki
	(00:37:10)	Uses LinkTool to create an object property
	(00:37:34)	Uses the DetailView to rename the property
	(00:38:04)	Interview
	(00:41:00)	Feedback
	(00:45:05)	End

B.5.4 TP2 (Jörg)

	Time	Event
	(00:00:00)	Consent form
	(00:01:08)	Introduction
	(00:02:29)	Background questionnaire
	(00:08:04)	Demo thinking aloud test
►	(00:09:27)	**Task 1**
	(00:12:50)	Creates an object property
	(00:13:10)	Opens the DetailView to change the name of the property
	(00:14:30)	Deletes the created object property
	(00:15:18)	Observes tools on top
	(00:15:45)	Notices button to show the class tree
	(00:16:02)	Notices QuickInfo
	(00:16:07)	Tries to find a visible class QuickSearch
	(00:16:13)	Is not sure how to hide the QuickSearch
	(00:16:19)	Finds a way to hide the QuickSearch
	(00:16:44)	Notices different node visualizations

(00:17:02) Finds a way to switch from icon mode to general mode
(00:17:04) Switches back to icon mode
(00:17:50) Finds the font scale switch
☺ (00:18:55) Control questions
► (00:19:30) **Task 2**
(00:19:38) Uses the QuickSearch
(00:19:56) Tries to use the QuickInfo tool to see the datatype properties and literals
(00:19:58) Switches to general mode
(00:20:20) Tries to change the node bounds
(00:20:32) Opens DetailView by accident
(00:21:00) Wants to change the node bounds
(00:21:36) Wants to use the magnifier to expand the bounds
☺ (00:22:22) Question if it is possible to see the datatype properties in the visualization
(00:22:50) Uncertain why classes have literals
► (00:23:20) **Task 3**
(00:23:30) Uses the QuickSearch to search target ontology root node
(00:23:50) Irritated due to overlapping nodes
(00:24:47) Notes that there are multiple subclasses
(00:26:04) Looks up the Inspector description
(00:26:50) Finds out how to use Inspector and Search Light
(00:28:55) Notes that he could found the related classes only from memorizing
► (00:29:50) **Task 4**
(00:30:05) Uses new action from the context menu to create a new resource
(00:30:17) Observes title of the DetailView
(00:30:30) Sees the create submenu but is confused about the title
(00:30:40) Uses the correct menu by random guess
(00:31:00) Tries edit the name by activating QuickEdit tool
(00:31:20) Opens the DetailView
(00:31:40) Changes to properties tab and opens the link tree
(00:32:45) Uncertain about the creation, looks into the Wiki
(00:33:16) Looks into the Wiki
(00:33:40) Uses the DatatypeProperty Creator
(00:35:10) Chooses the LiteralCreator by random guess
► (00:35:46) **Task 5**
(00:36:15) Memorizes the class Person and moves the created one
(00:37:10) Uses the create link submenu with wrong start node
(00:37:38) Tries to select both nodes and looks up the context menu
(00:38:15) Uses the context menu to set subject node
(00:39:09) Looks into the Wiki for object property creation
(00:40:05) Uses LinkTool to create an object property
(00:40:10) Chooses QuickEdit to change the name by random guess
(00:41:30) Interview
(00:49:00) Feedback
(00:53:25) End

B.5.5 TP3 (Harald)

Time	Event
(00:00:00)	Consent form

	(00:00:30)	Background questionnaire
	(00:05:30)	Introduction to the test
	(00:07:30)	Demo thinking aloud test
►	(00:08:10)	**Task 1**
	(00:09:15)	Wants to know what the buttons on the left part are good for
	(00:09:40)	Finds out how to zoom
	(00:10:02)	Notes it is cumbersome to move the viewport using scrollbars
	(00:10:14)	Notes good selection color
	(00:10:15)	Confused about crossing lines behind the node
	(00:10:30)	Opens DetailView
	(00:11:20)	Reads context menu entries
	(00:12:00)	Saves new object, uncertain if saving succeeded due to missing feedback
	(00:12:30)	Opens StarVisualization but does not understand
	(00:13:20)	Wants to see focus & context using marquee selection
	(00:14:40)	Opens DetailView of six nodes, system responds slowly
	(00:15:05)	Confused if the system has a deadlock, since systems provides no feedback
	(00:15:28)	Wants to close the DetailView, system does not react immediately
	(00:16:10)	Feels lost in the DetailViews
☺	(00:16:40)	Control questions
►	(00:18:30)	**Task 2**
	(00:18:50)	Uses QuickSearch to find the class
	(00:19:20)	Wants to see information that he is looking at classes
	(00:19:40)	Looks into context menu to find datatype properties,
	(00:20:00)	Reads properties from the DetailView but is not sure if this are datatype properties
☺	(00:21:04)	Quesition for different node visualization
►	(00:22:20)	**Task 3**
	(00:22:50)	Uses QuickSearch to find the target ontology class, appreciates animation
	(00:23:30)	Zooms to a subclass
	(00:24:10)	Identifies wrong subclass due to the layout
	(00:24:30)	Finds colored edges useful
	(00:24:57)	Would like to zoom using the mouse
☺	(00:25:40)	Asks if there is an easier way to find subclasses
	(00:26:10)	Tries to find subclasses using the class tree palette
►	(00:27:10)	**Task 4**
	(00:27:30)	Selects the create item from context menu
ϟ	(00:28:00)	Creation of subclass failed
	(00:30:30)	Reopens the OntologyVisualization
	(00:30:50)	Subclass creation succeeded and confuses user
	(00:31:05)	Opens DetailView to rename class
	(00:31:30)	Uncertain how to create datatype property from detail view
	(00:31:50)	Does not associate the word range with datatype
	(00:32:34)	Selects an entry from the metadata tab to set the datatype
	(00:33:00)	Does not understand what a literal is
☺	(00:33:13)	Explains what a literal is
	(00:33:50)	Enters description into DetailView by random guess
ϟ	(00:34:10)	Unexpected error
►	(00:35:05)	**Task 5**
	(00:35:10)	Uses QuickSearch to find class Person
	(00:35:30)	Wants to drag class Mitarbeiter on top of class Person to

		create the relationship
	(00:37:00)	Chooses the set start node action from context menu by random guess
	(00:37:25)	Selects create link action from context menu
	(00:37:55)	Wants to delete wrong edge using the keyboard
	(00:38:40)	Uncertain how to create the link, can not remember how the edge had been created before
	(00:39:50)	Deletes created class but wants to delete link
	(00:34:02)	Interview
	(00:43:00)	Feedback
	(00:51:00)	End

B.5.6 TP4 (Robert)

	Time	Event
	(00:00:00)	Consent form
	(00:02:10)	Background questionnaire
	(00:06:33)	Introduction
	(00:07:40)	Demo thinking aloud test
►	(00:11:43)	**Task 1**
	(00:12:09)	Complains about slow DetailView
	(00:13:09)	Wonders about nodes that lie over each other
	(00:13:45)	Finds it uncommon that right mouse button is used to move the viewport
	(00:14:44)	Reads the DetailView
	(00:15:10)	Uncertain about the datatype properties available
	(00:15:50)	Wants to see a tree for the types properties
	(00:18:00)	Uses double click to open DetailView
	(00:18:50)	Clicks StarVisualization and tries to interpret object property
	(00:20:00)	Double clicks to open DetailView again in StarVisualization
☺	(00:21:58)	Control questions
►	(00:26:40)	**Task 2**
	(00:27:00)	Tries to use the global search field
	(00:27:45)	Finds Navigator, tries to hide again
	(00:28:45)	Finds QuickSearch and wants to have the search field in the toolbar
	(00:28:45)	Happy about automatic centering
	(00:29:55)	Tries to find datatype properties using the DetailView
	(00:30:34)	Finds tooltips in the DetailView
	(00:32:50)	Uses subtype visualization
ϟ	(00:33:10)	Unexpected error
	(00:33:30)	Looks into context menu to switch to general node visualization
	(00:33:40)	Expects to switch visualization in the menu
	(00:34:50)	Tries to use the class tree to find datatype properties
	(00:35:20)	Finds button icon switch
►	(00:37:00)	**Task 3**
	(00:37:29)	Opens subtype visualization
	(00:37:44)	Opens StarVisualization
	(00:39:00)	Confused about green node (marked from previous search)
	(00:39:30)	Remembers previous search and clears result
	(00:39:50)	Moves node to see the related nodes
	(00:42:00)	Finds Search Light

	(00:42:10)	Reads Statebar tooltips
	(00:42:58)	Uses Magnifier to read the related node titles
	(00:43:40)	Uses QuickSearch to find the class Person
	(00:44:10)	Types and presses enter, QuickSearch has no focus, DetailView opens by accident
	(00:44:30)	Wants to search for multiple classes at once
	(00:46:10)	Wants to have multiple colors for multiple selections
►	(00:46:30)	**Task 4**
	(00:47:00)	Wants to clear selection by clicking an empty area
	(00:47:30)	Uses new action from the context menu to create a new resource
	(00:48:00)	Uses create submenu
	(00:48:11)	Uses DetailView to rename the class
	(00:48:31)	Creates datatype property, uncertain about domain setting
	(00:49:00)	Uncertain how to set the range
	(00:50:10)	Does not know difference between range and range value type
	(00:50:40)	Tries to use keyboard to remove range setting
☺	(00:51:30)	Interrupts task, asks for alternative editing method
	(00:52:20)	Uncertain how to start editing using the tool
	(00:52:30)	Starts editing using QuickEdit, notes that it is way easier to use than the DetailView
↯	(00:52:40)	Datatype property is not created
↯	(00:54:40)	Literal is not created, user does not see the relationship in the DetailView
►	(00:55:30)	**Task 5**
	(00:55:40)	Uses QuickSearch to find the class Person
	(00:56:00)	Opens DetailView to create subclass relation
☺	(00:57:28)	Shows how to create a new subclass due to the bug
↯	(00:57:28)	Class creation not possible
	(00:58:30)	Uncertain if subclass relations are object properties
	(00:59:00)	Uses double click to set start and end node
☺	(01:00:30)	Question for possibility to edit resources
	(01:01:00)	Finds QuickEdit
	(01:00:00)	Interview
	(01:05:00)	Feedback
	(01:09:18)	End

B.5.7 TP5 (Markus)

	Time	Event
	(00:00:00)	Introduction
	(00:00:50)	Consent form
	(00:01:29)	Background questionnaire
	(00:05:09)	Demo thinking aloud test
►	(00:06:48)	**Task 1**
	(00:07:20)	Finds pan
	(00:07:37)	Finds zoom
	(00:08:10)	Reads tooltips of the buttons in the toolbar
	(00:08:40)	Switches between icon and general mode
	(00:08:51)	Reads tooltips of in the Statebar
	(00:09:36)	Wants to finish task 1
☺	(00:09:39)	Control questions
►	(00:11:40)	**Task 2**

	(00:11:50)	Uses zoom and pan to search
☺	(00:12:20)	Question if there is another way to search
	(00:12:30)	Uses QuickSearch
►	(00:12:48)	**Task 3**
	(00:13:05)	Uses QuickSearch to find the class
	(00:13:14)	Double clicks to select search result
	(00:13:35)	Moves the nodes to see subclasses
	(00:13:46)	Follows a subclass edge using pan
	(00:14:10)	Opens a DetailView while reading without taking notice
	(00:16:08)	Moves the target ontology node to see the subclass edges
☺	(00:17:55)	Question if easier way to find subclasses
	(00:18:00)	Looks at the Statebar, switches to icon mode
	(00:18:37)	Finds Search Light
►	(00:18:45)	**Task 4**
	(00:19:20)	Uses new action from the context menu to create new resource
	(00:20:40)	Opens the link MultiTree but closes again since, seems to be confused about its meaning
	(00:20:56)	Creates a new datatype property using the DetailView
	(00:21:30)	Closes DetailView
	(00:21:40)	Uses QuickSearch since the class created is not visible in the visualization
	(00:21:45)	Opens the DetailView again
☺	(00:22:00)	Question what went wrong
	(00:22:30)	Uncertain about datatype property range setting
	(00:22:49)	Looks into Wiki about datatype property creation
☺	(00:23:10)	Question if another way to create something is available
	(00:23:11)	Reads the tooltips of available tools
	(00:24:27)	Chooses create submenu by random guess
	(00:24:41)	Opens DetailView to rename the class
	(00:24:48)	Tries to delete the node using keyboard, uncertain about subclass relation of class Zielontologie
☺	(00:25:00)	Question what went wrong
	(00:25:18)	Creates the subclass again
	(00:25:40)	Tries to rename and add something to the class using the context menu
	(00:26:00)	Tries to start editing using datatype creator
	(00:27:00)	Tries to add literal using datatype creator
	(00:29:20)	Feels unable to create a literal
►	(00:29:40)	**Task 5**
	(00:30:20)	Uses QuickSearch to find the class Person
	(00:30:29)	Notices LiteralCreator
	(00:30:35)	Selects LinkTool, tries to drag the created class on top of class Person
	(00:31:20)	Creates a new object property
	(00:33:47)	Accidentally selectes wrong property to rename class
	(00:34:02)	Interview
	(00:37:00)	Feedback
	(00:44:30)	End

B.5.8 TP6 (Ronald)

Time	Event
(00:00:00)	Consent form

(00:01:42) Introduction
(00:02:16) Background questionnaire
(00:05:25) Demo thinking aloud test
► (00:08:05) **Task 1**
(00:08:43) Knows how to zoom
(00:08:54) Remembers how to pan
(00:09:05) Notices that nodes can lie on top on each other
(00:09:30) Uncertain about the search on top
(00:09:40) Reads toolbar tooltips
(00:09:55) Finds QuickSearch
(00:10:20) Tries to duplicate a node
(00:10:34) Uncertain if closing DetailView closes visualization as well
(00:10:20) Tries to duplicate a node
(00:10:50) Finds QuickInfo
(00:11:00) Finds class tree palette, uses coloring
(00:12:00) Finds creation tools and noticed their keyboard shortcuts
(00:12:30) Finds mode switch button
(00:12:40) Wants to resize the node using keyboard and mouse
(00:13:20) Activates and deactivates coordinator without seeing effect; can not imagine its purpose
(00:13:50) Activates and deactivates title scaling
(00:14:10) Titles too small to read
(00:14:40) Activates Inspector, first try fails due to tooltip
(00:15:30) Understands purpose of Search Light
(00:15:44) Finds the Magnifier
(00:15:51) Tries to click the selection indicator field of the Statebar
☺ (00:16:55) Control questions
► (00:19:21) **Task 2**
(00:19:21) Uses QuickSearch and explains its features
(00:20:31) Opens DetailView to find datatype properties since node visualization is too small
(00:21:00) Explains DetailView features
☺ (00:22:25) Question if it is possible to name the datatype properties directly from the visualization
(00:22:50) Uses property tab from the DetailView to name the properties
(00:23:30) Notices line in the general node visualization
(00:24:20) Uncertain about the term literal
► (00:24:30) **Task 3**
(00:25:20) Tries to activate the Search Light to find related classes, fails due to overlapping tooltip
(00:25:40) Moves the node to see edges
(00:25:55) Notes that lines have similar color
(00:26:52) Search Light activation succeeded
(00:27:09) Activates title scaling
☺ (00:29:35) Question to name object properties of the class Person and the subclasses
(00:30:00) Uses zoom to zoom to some related edges
(00:30:30) Wants to use Inspector, activation fails due to tooltip overlay
(00:31:20) Uses Inspector title overlay
(00:31:39) Wonders about the blue circle of the inspector which had never been visible before
► (00:32:00) **Task 4**
(00:32:50) Uses create action from the context menu
(00:33:04) Opens DetailView to rename the class
(00:33:25) Uses new action from the properties tab to create a new datatype property
(00:33:50) Uncertain how to set range

(00:34:50) Chooses new action to create a new range value type from DetailView
(00:35:50) Chooses link action by random guess from context menu
(00:36:10) Uncertain how to save link setting in the MultiTree
(00:37:00) Can not see difference between range and range value type
(00:39:00) Notes slow sorting of the MultiTree
► (00:40:00) Question if there is an easier way to create datatype properties
(00:40:10) Notices DatatypeProperty Creator
► (00:41:20) **Task 5**
(00:42:38) Would not use object property creation tool to create a subclass
(00:44:10) Fails to set the superclass relationship using the MultiTree
(00:45:05) Succeeds to set the superclass relation ship using the MultiTree
(00:45:24) Confused since the visualization does not show superclass but direct superclass
(00:47:00) Uses LinkTool to create the object property
(00:47:15) Opens DetailView to rename the property
(00:47:56) Opens DetailView accidentally for the same property
(00:48:00) Sets start node for link operation by accident
(00:49:03) Unclear which node needs to be selected at first for subclass
ϟ (00:49:10) Direct subclass can not be created anymore
since subclass exists, user is confused
(00:49:50) Interview
(01:03:50) Feedback
(01:12:48) End

Bibliography

Abello, James, Schulz, Hans-Jörg, Schumann, Heidrun, and Tominski, Christian [2007]. *CGV - Coordinated Graph Visualization Interactive Poster*. In *Proc. 13th IEEE Symposium on Information Visualization 2007 (InfoVis'07)*. IEEE Computer Society. http://www.informatik.uni-rostock.de/~schumann/papers/2006+/infovis07_ct.pdf. (Cited on pages 22 and 23.)

Agrawala, Maneesh and Heer, Jeffrey [2006]. *Software Design Patterns for Information Visualization*. *IEEE Transactions on Visualization and Computer Graphics*, 12(5), pages 853–860. ISSN 1077-2626. doi:10.1109/TVCG.2006.178 . (Cited on page 59.)

Andrews, Keith [2002]. *Visualising Information Structures, Aspects of Information Visualisation*. ftp://ftp.iicm.edu/pub/keith/habil/visinfo.pdf. Habilitation, visited on 23 Apr 2007. (Cited on page 21.)

Bederson, Benjamin B., Grosjean, Jesse, and Meyer, Jon [2004]. *Toolkit Design for Interactive Structured Graphics*. *IEEE Transactions Software Engineering*, 30(8), pages 535–546. ISSN 0098-5589. doi:10.1109/TSE.2004.44 . (Cited on pages 29 and 33.)

Bederson, Benjamin B. and Hollan, James D. [1994]. *Pad++: A Zooming Graphical Interface for Exploring Alternate Interface Physics*. In *Proc. 7th Annual ACM Symposium on User Interface Software and Technology (UIST '94)*, pages 17–26. ACM, New York, NY, USA. ISBN 0897916573. doi:10.1145/192426.192435 . (Cited on page 29.)

Benson, David [2006]. *JGraph and JGraph Layout Pro User Manual, For JGraph Version 5.9.1.0 and JGraph Layout Pro 1.3.0.9*. http://www.jgraph.com/. Visited on 19 Mar 2009. (Cited on page 59.)

Bier, Eric A., Stone, Maureen C., Pier, Ken, Buxton, William, and DeRose, Tony D. [1993]. *Toolglass and Magic Lenses: The See-Through Interface*. In *Proc. 20th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '93)*, pages 73–80. ACM, New York, NY, USA. ISBN 0897916018. doi:10.1145/166117.166126 . (Cited on page 25.)

Bobrow, Daniel G., Mittal, Sanjay, and Stefik, Mark J. [1986]. *Expert Systems: Perils and Promise*. *Communications of the ACM*, 29(9), pages 880–894. ISSN 0001-0782. doi:10.1145/6592.6597 . (Cited on page 3.)

Boren, Ted and Ramey, Judith [2000]. *Thinking Aloud: Reconciling Theory and Practice*. *IEEE Transactions on Professional Communication*, 43(3), pages 261–278. ISSN 0361-1434. doi:10.1109/47.867942 . (Cited on page 67.)

Brickley, Dan and Guha, Ramanathan V. [2004]. *RDF Vocabulary Description Language 1.0: RDF Schema W3C Recommendation 10 February 2004*. Technical Report, W3C. http://www.w3.org/TR/2004/REC-rdf-schema-20040210/. Visited on 08 Apr 2007. (Cited on page 8.)

Canas, Alberto J., Carff, Roger, Hill, Greg, Carvalho, Marco, Arguedas, Marco, Eskridge, Thomas C., Lott, James, and Carvajal, Rodrigo [2005]. *Concept Maps: Integrating Knowledge and Information Visualization*. In *Knowledge and Information Visualization*. Springer Berlin / Heidelberg. ISBN 3540269212. doi:10.1007/11510154_11 . (Cited on page 4.)

Card, Stuart K., Mackinlay, Jock D., and Shneiderman, Ben (Editors) [1999]. *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. ISBN 1558605339. (Cited on page 23.)

Chen, Ing-Xiang, Fan, Chun-Lin, Lo, Pang-Hsiang, Kuo, Li-Chia, and Yang, Cheng-Zen [2005]. *ISWIVE: An Integrated Semantic Web Interactive Visualization Environment*. In *19th International Conference on Advanced Information Networking and Applications (AINA 2005)*, volume 2, pages 701–706. doi:10.1109/AINA.2005.228 . (Cited on page 31.)

Cooper, Allan [1999]. *The Inmates Are Running The Asylum: Why High Tech Products Drive Us Crazy And How To Restore The Sanity*. SAMS Publishing. ISBN 0672326140. (Cited on page 41.)

Eklund, Peter, Becker, Peter, and Roberts, Nataliya [2006]. *RDF-based Peer-To-Peer Based Ontology Editing*. *Journal of Digital Information Management*, 4(1), pages 50–55. doi:10.1.1.20.6809 . `http://www.kvocentral.org/kvopapers/eklund-Peer-to-Peer.pdf`. (Cited on pages 35 and 36.)

Eklund, Peter, Roberts, Nataliya, and Green, Steve [2002]. *Ontorama: Browsing an RDF Ontology Using a Hyperbolic-Like Browser*. In *Proc. First International Symposium on Cyber Worlds (CW'02)*, pages 405–411. IEEE Computer Society, Washington, DC, USA. ISBN 0769518621. (Cited on pages 35 and 36.)

Feigenbaum, Edward A., Buchanan, Bruce G., and Lederberg, Joshua [1970]. *On Generality and Problem Solving: A Case Study Using the DENDRAL Program*. Technical Report, Computer Science Department, Stanford University, Stanford, CA, USA. `http://profiles.nlm.nih.gov/BB/A/B/K/V/`. Visited on 01 Aug 2007. (Cited on page 3.)

Fekete, Jean-Daniel and Plaisant, Catherine [1999]. *Excentric Labeling: Dynamic Neighborhood Labeling for Data Visualization*. In *Proc. SIGCHI Conference on Human Factors in Computing Systems (CHI '99)*, pages 512–519. ACM, New York, NY, USA. ISBN 0201485591. doi:10.1145/302979.303148 . (Cited on pages 25, 26, 43, 65 and 66.)

Franconi, Enrico [1996]. *Lecture Notes 'Description Logics'*. `http://www.inf.unibz.it/~franconi/dl/course/`. Visited on 15 Oct 2008. (Cited on page 3.)

Furnas, George W. [1986]. *Generalized Fisheye Views*. In *Proc. SIGCHI Conference on Human Factors in Computing Systems (CHI '86)*, pages 16–23. ACM, New York, NY, USA. ISBN 0897911806. doi:10.1145/22627.22342 . (Cited on page 24.)

Furnas, George W. [2006]. *A Fisheye Follow-Up: Further Reflections on Focus + Context*. In *Proc. SIGCHI Conference on Human Factors in Computing Systems (CHI '06)*, pages 999–1008. ACM, New York, NY, USA. ISBN 1595933727. doi:10.1145/1124772.1124921 . (Cited on page 24.)

Gansner, Emden R., Koren, Yehuda, and North, Stephen C. [2005]. *Topological Fisheye Views for Visualizing Large Graphs*. *IEEE Transactions on Visualization and Computer Graphics*, 11(4), pages 457–468. ISSN 1077-2626. doi:10.1109/TVCG.2005.66 . (Cited on page 24.)

Goyal, Sunil and Westenthaler, Rupert [2004]. *RDF Gravity (RDF Graph Visualization Tool)*. `http://semweb.salzburgresearch.at/apps/rdf-gravity/index.html`. Visited on 15 Mar 2009. (Cited on page 31.)

Gruber, Thomas R. [1993]. *A Translation Approach to Portable Ontology Specifications*. *Knowl. Acquis.*, 5(2), pages 199–220. ISSN 1042-8143. doi:10.1006/knac.1993.1008 . http://ksl-web.stanford.edu/KSL_Abstracts/KSL-92-71.html. (Cited on page 6.)

Guarino, Nicola [1998]. *Formal Ontology and Information Systems*. In *Proc. 1st International Conference on Formal Ontologies in Information Systems, Formal Ontology and Information Systems (FOIS '98)*, pages 3–15. IOS Press. http://citeseer.ist.psu.edu/guarino98formal.html. (Cited on page 7.)

Guarino, Nicola and Poli, Roberto [1995]. *Formal Ontology in Conceptual Analysis and Knowledge Representation*. *International Journal of Human and Computer Studies (Special Issue)*, 43, Issues 5–6, pages 625–640. doi:10.1006/ijhc.1995.1066 . http://citeseer.ist.psu.edu/guarino95formal.html. (Cited on page 7.)

Hauser, Helwig [2005]. *Lecture Notes 'Visualization' Technical University of Vienna*. http://www.cg.tuwien.ac.at/courses/Visualisierung/. Visted on 01 July 2007. (Cited on page 23.)

Hayes, Patrick and McBride, Brian [2004]. *RDF Semantics, W3C Recommendation 10 February 2004*. W3C recommendation, W3C. http://www.w3.org/TR/2004/REC-rdf-mt-20040210/. Visited on 07 Apr 2007. (Cited on page 8.)

Hayes-Roth, Frederick and Jacobstein, Neil [1994]. *The State of Knowledge-Based Systems*. *Communications of the ACM*, 37(3), pages 26–39. ISSN 0001-0782. doi:10.1145/175247.175249 . (Cited on page 3.)

Healey, Christopher G. [2007]. *Perception in Visualization*. http://www.csc.ncsu.edu/faculty/healey/PP/index.html. Visited on 28 Jan 2009. (Cited on page 22.)

Healey, Christopher G., Booth, Kellogg S., and Enns, James T. [1995]. *Visualizing Real-Time Multivariate Data Using Preattentive Processing*. *ACM Transactions on Modeling and Computer Simulation*, 5(3), pages 190–221. ISSN 1049-3301. doi:10.1145/217853.217855 . (Cited on page 22.)

Hearst, Martin [2004]. *Information Visualization: Principles, Promise, and Pragmatics (CHI '04 Tutorial Notes)*. http://www.sigchi.org/chi2003/tutorial-details.html. Visited on 01 Aug 2008. (Cited on page 21.)

Herman, Ivan, Melançon, Guy, and Marshall, Scott [2000]. *Graph Visualization and Navigation in Information Visualization: A Survey*. *IEEE Transactions on Visualization and Computer Graphics*, 6(1), pages 24–43. doi:10.1109/2945.841119 . http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.28.8892. (Cited on pages 21 and 26.)

Janecek, Paul [2004]. *Interactive Semantic Fisheye Views for Information Workspaces*. PhD Thesis, Ecole Polytechnique Federale De Lausanne. http://library.epfl.ch/theses/?nr=2994. (Cited on page 24.)

Janecek, Paul and Pu, Pearl [2002]. *A Framework for Designing Fisheye Views to Support Multiple Semantic Contexts*. In *Proc. International Conference on Advanced Visual Interfaces (AVI '02)*, pages 51–58. ACM. http://hci.epfl.ch/publications/2002/AVI_final_bw.pdf. (Cited on page 24.)

Johnson, Brian and Shneiderman, Ben [1991]. *Tree-Maps: A Space-Filling Approach To The Visualization Of Hierarchical Information Structures*. In *Proc. 2nd Conference on Visualization '91 (VIS '91)*, pages 284–291. IEEE Computer Society Press, Los Alamitos, CA, USA. ISBN 0818622458. (Cited on page 27.)

Johnson, Jeff [2008]. *GUI Bloopers 2.0*. Morgan Kaufmann Publishers. ISBN 0123706432. (Cited on pages 40 and 44.)

Katifori, Akrivi, Halatsis, Constantin, Lepouras, George, Vassilakis, Costas, and Giannopoulou, Eugenia [2007]. *Ontology Visualization Methods – A Survey*. *ACM Computing Surveys*, 39(4), page 10. ISSN 0360-0300. doi:10.1145/1287620.1287621 . (Cited on page 31.)

Keim, Daniel [2002]. *Information Visualization and Visual Data Mining*. *IEEE Transactions on Visualization and Computer Graphics*, 8(1), pages 1–8. doi:10.1109/2945.981847 . (Cited on page 22.)

Klyne, Graham, Carroll, Jeremy J., and McBride, Biran [2004]. *Resource Description Framework (RDF): Concepts and Abstract Syntax*. W3C recommendation, W3C. http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/. Visited on 01 Aug 2008. (Cited on page 8.)

Kosara, Robert, Miksch, Silvia, Hauser, Helwig, Schrammel, Johann, Giller, Verena, and Tscheligi, Manfred [2002]. *Useful Properties of Semantic Depth of Field for Better F+C Visualization*. In *Proc. Symposium on Data Visualisation 2002 (VISSYM '02)*, pages 205–210. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland. ISBN 158113536X. (Cited on pages 25 and 26.)

Kosslyn, Stephen M. [1998]. *Understanding Charts and Graphs. Applied Cognitive Psychology*, 3(3), pages 185–225. doi:10.1002/acp.2350030302 . http://www.wjh.harvard.edu/~kwn/KosslynArticles.html. (Cited on page 30.)

Krahmer, Emiel and Ummelen, Nicole [2004]. *Thinking About Thinking Aloud: A Comparison of Two Verbal Protocols for Usability Testing*. *IEEE Transactions on Professional Communication*, 47(2), pages 105–117. ISSN 0361-1434. doi:10.1109/TPC.2004.828205 . (Cited on page 67.)

Kreuzer, Martin and Kühling, Stefan [2006]. *Logik für Informatiker*. Pearson Studium. ISBN 3827372154. (Cited on page 3.)

Lamping, John, Rao, Ramana, and Pirolli, Peter [1995]. *A Focus+Context Technique Based on Hyperbolic Geometry for Visualizing Large Hierarchies*. In *Proc. SIGCHI Conference on Human Factors in Computing Systems (CHI '95)*, pages 401–408. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA. ISBN 0201847051. doi:10.1145/223904.223956 . (Cited on pages 24, 25 and 30.)

Lassila, Ora and McGuinness, Deborah [2001]. *The Role of Frame-Based Representation On The Semantic Web*. Nr. ksl-01-02, Knowledge Systems Laboratory, Stanford University. http://ksl.stanford.edu/KSL_Abstracts/KSL-01-02.html. Visited on 28 Jan 2009. (Cited on page 8.)

Lau, Keith, Rensink, Ronald A., and Munzner, Tamara [2004]. *Perceptual Invariance of Nonlinear Focus+Context Transformations*. In *Proc. 1st Symposium on Applied Perception in Graphics and Visualization (APGV '04)*, pages 65–72. ACM, New York, NY, USA. ISBN 1581139144. doi:10.1145/1012551.1012563 . (Cited on page 29.)

Lederberg, Joshua [1987]. *How DENDRAL was Conceived and Born*. In *Proc. of ACM Conference on History of Medical Informatics*, pages 5–19. ACM, New York, NY, USA. ISBN 0897912489. doi:10.1145/41526.41528 . (Cited on page 3.)

Lee, Tim Berners [1999]. *Weaving the Web: Origins and Future of the World Wide Web*. Orion, London. ISBN 0752820903. (Cited on page 8.)

Lee, Tim Berners, Connolly, Dan, and Hawke, Sandro [2003]. *Semantic Web Tutorial Using N3*. http://www.w3.org/2000/10/swap/Primer. Tutorial notes, 12th International World Wide Web Conference (WWW2003), visited on 26 Nov 2008. (Cited on page 8.)

Leung, Ying K. and Apperley, Mark D. [1994]. *A Review and Taxonomy of Distortion-Oriented Presentation Techniques*. *ACM Transactions on Computer-Human Interaction*, 1(2), pages 126–160. ISSN 1073-0516. doi:10.1145/180171.180173 . (Cited on page 23.)

Liebig, Thorsten [2003]. *OntoTrack: Fast browsing and Easy Editing of Large Ontologies*. In *Proc. 2nd International Workshop on Evaluation of Ontology-Based Tools (EON2003), located at ISWC03*. Sanibel Island, USA. http://www.informatik.uni-ulm.de/ki/Liebig/papers/LiNo03.pdf. (Cited on page 34.)

Liebig, Thorsten and Noppens, Olaf [2004]. *OntoTrack: Combining Browsing and Editing with Reasoning and Explaining for OWL Lite Ontologies*. In *Proc. 3rd International Semantic Web Conference (ISWC2004)*, pages 244–257. Number 3298 in Lecture Notes in Computer Science, Springer Verlag, Hiroshima, Japan. doi:10.1007/b102467 . http://www.informatik.uni-ulm.de/ki/Liebig/papers/liebig-et-al-iswc04.pdf. (Cited on pages 34 and 35.)

Lokuge, Ishantha and Ishizaki, Suguru [1995]. *GeoSpace: An Interactive Visualization System for Exploring Complex Information Spaces*. In *Proc. SIGCHI Conference on Human Factors in Computing Systems (CHI '95)*, pages 409–414. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA. ISBN 0201847051. doi:10.1145/223904.223959 . (Cited on pages 26 and 27.)

Mackinlay, Jock D., Robertson, George G., and Card, Stuart K. [1991]. *The Perspective Wall: Detail and Context Smoothly Integrated*. In *Proc. SIGCHI Conference on Human Factors in Computing Systems (CHI '91)*, pages 173–176. ACM, New York, NY, USA. ISBN 0897913833. doi:10.1145/108844.108870 . (Cited on page 24.)

Manola, Frank and Miller, Eric [2004]. *RDF Primer, W3C Recommendation 10 February 2004*. Technical Report, W3C. http://www.w3.org/TR/rdf-primer/. Visited on 04 Aug 2007. (Cited on pages 8, 9 and 31.)

McGuinness, Deborah and Harmelen, Frank [2004]. *OWL Web Ontology Language Overview*. Technical Report, W3C. http://www.w3.org/TR/owl-features/. Visited on 04 Apr 2008. (Cited on page 8.)

Minsky, Marvin [1974]. *A Framework for Representing Knowledge*. Memo, MIT-AI Laboratory. http://web.media.mit.edu/~minsky/papers/Frames/frames.html. Visited on 06 Nov 2008. (Cited on page 6.)

Nebel, Bernhard and Smolka, Gert [1992]. *Attributive Description Formalisms And The Rest Of The World*. In *Text Understanding in LILOG*, pages 439–452. Springer Berlin / Heidelberg. ISBN 3540545941. doi:10.1007/3-540-54594-8_74 . (Cited on page 8.)

Norman, Donald [1987]. *Some Observations on Mental Models*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. ISBN 0934613249, 241–244 pages. (Cited on page 44.)

Norman, Donald [2005]. *Human-Centered Design Considered Harmful*. *Interactions*, 12(4), pages 14–19. ISSN 1072-5520. doi:10.1145/1070960.1070976 . (Cited on page 44.)

Novak, Joseph and Canas, Alberto [2008]. *The Theory Underlying Concept Maps and How to Construct Them*. Technical report, Institute for Human and Machine Cognition (IHMC). http://cmap.ihmc.us/Publications/ResearchPapers/TheoryUnderlyingConceptMaps.pdf. Rev 2008-01, visted on 28.11.2008. (Cited on pages 4 and 5.)

Noy, Natalya, Fergerson, Ray, and Musen, Mark [2000]. *The Knowledge Model of Protégé-2000: Combining Interoperability and Flexibility*. In *Proc. 12th European Workshop on Knowledge Acquisition, Modeling and Management (EKAW '00)*, pages 17–32. Springer-Verlag, London, UK. ISBN 3540411194. (Cited on page 32.)

OntoTrackManual [2009]. *OntoTrack: Fast Browsing and Easy Editing of Large Ontologies*. http://www.informatik.uni-ulm.de/ki/ontotrack/. Visited on 28 Jan 2009. (Cited on page 34.)

Perlin, Ken and Fox, David [1993]. *Pad: An Alternative Approach to the Computer Interface*. In *Proc. 20th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '93)*, pages 57–64. ACM, New York, NY, USA. ISBN 0897916018. doi:10.1145/166117.166125 . (Cited on page 29.)

Pietriga, Emmanuel [2005]. *A Toolkit for Addressing HCI Issues in Visual Language Environments*. *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 00, pages 145–152. doi:10.1109/VLHCC.2005.11 . (Cited on pages 31 and 32.)

Plaisant, Catherine, Grosjean, Jesse, and Bederson, Benjamin B. [2002]. *SpaceTree: Supporting Exploration in Large Node Link Tree, Design Evolution and Empirical Evaluation*. In *Proc. IEEE Symposium on Information Visualization (InfoVis'02)*, pages 57–64. IEEE Press. doi:10.1109/INFVIS.2002.1173148 . (Cited on pages 27 and 29.)

Quillian, Michael R. [1969]. *The Teachable Language Comprehender: A Simulation Program And Theory Of Language*. *Communications of the ACM*, 12(8), pages 459–476. ISSN 0001-0782. doi:10.1145/363196.363214 . (Cited on page 6.)

Rao, Ramana and Card, Stuart K. [1994]. *The Table Lens: Merging Graphical and Symbolic Representations in an Interactive Focus+Context Visualization for Tabular Information.* In *Proc. of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 318–322. ACM, New York, NY, USA. ISBN 0897916506. doi:10.1145/191666.191776 . (Cited on page 24.)

Reimer, Ulrich [1998]. *Vom Knowledge Engineering zum Knowledge Management: Entwurf, Aufbau und Wartung von Wissensdatenbanken*. In *Proc. GI-Workshop, Münster, 11.-13. März 1998*, volume 9, pages 1–5. M. Jeusfeld c/o Redaktion Sun SITE, Informatik V, RWTH Aachen. ISSN 1613-0073. http://CEUR-WS.org/Vol-9/. (Cited on page 4.)

Russel, Stuart and Norvig, Peter [2004]. *Künstliche Intelligenz, Ein moderner Ansatz*. 2nd Edition. Pearson Education Deutschland. ISBN 3827370892. (Cited on page 3.)

Sarkar, Manojit and Brown, Marc H. [1992]. *Graphical Fisheye Views of Graphs*. In *Proc. SIGCHI Conference on Human Factors in Computing Systems (CHI '92)*, pages 83–91. ACM, New York, NY, USA. ISBN 0897915135. doi:10.1145/142750.142763 . (Cited on page 23.)

Schmolze, James G. and Lipkis, Thomas A. [1985]. *An Overview Of The KL-ONE Knowledge Representation System*. *Cognitive Science*, 9, pages 171–216. http://nlp.shef.ac.uk/kr/papers/klone.ps. (Cited on page 6.)

Shneiderman, Ben [1983]. *Direct Manipulation: A Step Beyond Programming Languages*. *Computer*, 16(8), pages 57–69. ISSN 0018-9162. doi:10.1109/MC.1983.1654471 . (Cited on page 23.)

Shneiderman, Ben [1996]. *The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations*. In *Proc. 1996 IEEE Symposium on Visual Languages (VL '96)*, pages 336–343. IEEE Computer Society, Washington, DC, USA. ISBN 081867508X. doi:10.1109/VL.1996.545307 . (Cited on pages 21 and 23.)

Smith, Barry [2004]. *Ontology*. In *The Blackwell Guide to the Philosophy of Computing and Information*. Blackwell Publishing Ltd. ISBN 0631229191. (Cited on page 6.)

Smith, Barry, Ceusters, Werner, and Temmerman, Rita [2005]. *Wüsteria*. In *Connecting Medical Informatics and Bio-Informatics: Proc. of MIE2005 - The XIXth International Congress of the European Federation for Medical Informatics*, *Studies in Health Technology and Informatics*, volume

116/2005, pages 647–652. IOS Press. ISBN 1586035495. http://ontology.buffalo.edu/medo/Wuesteria.pdf. (Cited on page 7.)

Smith, Michael K., Welty, Chris, and McGuinness, Deborah [2004]. *OWL Web Ontology Language Guide*. Technical Report, W3C. http://www.w3.org/TR/2004/REC-owl-guide-20040210/. Visited on 15 June 2008. (Cited on page 8.)

Sowa, John F. [1979]. *Semantics of Conceptual Graphs*. In *Proc. 17th Annual Meeting on Association for Computational Linguistics*, pages 39–44. Association for Computational Linguistics, Morristown, NJ, USA. doi:10.3115/982163.982175 . http://www.aclweb.org/anthology-new/P/P79/P79-1010.pdf. (Cited on pages 4 and 5.)

Sowa, John F. [1992]. *Semantic Networks*. In Shapiro, Stuart C. (Editor), *Encyclopedia of Artificial Intelligence*, 2nd Edition. Wiley. ISBN 0471503071. http://www.jfsowa.com/pubs/semnet.htm. Cited revised online version, visited on 20 Nov 2008. (Cited on pages 4 and 7.)

Spence, Robert [2000]. *Information Visualization*. 1st Edition. Addison Wesley. ISBN 0201596261. (Cited on page 21.)

Storey, Margaret-Anne, Musen, Mark, Silva, John, Ernst, Neil, Fergerson, Ray, and Noy, Natasha [2001]. *Jambalaya: Interactive Visualization to Enhance Ontology Authoring and Knowledge Acquisition*. In *Protégé Workshop on Interactive Tools for Knowledge Capture (K-CAP-2001)*. (Cited on page 32.)

Storey, Margaret-Anne, Noy, Natasha, Musen, Mark, Best, Casey, Fergerson, Ray, and Ernst, Neil [2002]. *Jambalaya: An Interactive Environment for Exploring Ontologies*. In *Proc. 7th International Conference on Intelligent User Interfaces (IUI '02)*, pages 239–239. ACM, New York, NY, USA. ISBN 1581134592. doi:10.1145/502716.502778 . (Cited on page 32.)

Sutcliffe, Alistrair [2002]. *User-Centered Requirements Engineering (Theory And Practice)*. Springer-Verlag London Berlin Heidelberg. ISBN 1852335173. (Cited on page 39.)

Ware, Colin [2004]. *Information Visualization, Perception for Design*. 2nd Edition. Morgan Kaufmann. ISBN 1558608192. (Cited on page 22.)

Wattenberg, Martin [2006]. *Visual Exploration of Multivariate Graphs*. In *Proc. SIGCHI Conference on Human Factors in Computing Cystems (CHI '06)*, pages 811–819. ACM, New York, NY, USA. ISBN 1595933727. doi:10.1145/1124772.1124891 . (Cited on pages 27 and 28.)

Zakhour, Sharon, Hommel, Scott, Royal, Jacob, Rabinovitch, Isaac, Risser, Tom, and Hoeber, Mark [2009]. *The Java Tutorial: A Short Course on the Basics (Paperback)*. 4th Edition. Prentice Hall. ISBN 0321334205. http://java.sun.com/docs/books/tutorial/. (Cited on page 48.)