Project Semantic Web for Pathology
Report AP2

FU/NBI, Charite

April 18, 2005

Abstract

This report presents our achievements during the second work package of the project "Semantic Web in the Pathology", whose principal aim was to design and implement the knowledge component, which is the core component for the realization of the pathology retrieval system. The knowledge component manages pathology-relevant knowledge, represented as ontologies or rules, which is used both for retrieval and quality assurance purposes.

Chapter 1

Introduction: Main goals of AP2

The AP2 is intended to realize the knowledge component, whose main aim is the management of domain-related information, which is represented in a Semantic Web compatible form and is used both by application components like the retrieval engine and the quality assurance tool, and in text analysis procedures. The first and the third use cases are supported by domain-relevant ontological information, while the quality management is realized by means of rules describing important quality criteria for medical reports and diagnostic procedures.

Typical engineering methodologies, which describe the subtasks to be performed when constructing knowledge bases were used for the implementation of the pathology knowledge base. Additionally a module for the management of the encoded domain knowledge and interfaces to future application components were implemented as extension of the tool resulted from AP1.

This report gives an account of our work towards the realization of these goals. Section 2 describes the implementation of the pathology ontology library presenting the alternatives we considered for this purpose and lessons learned. Section 3 follows the same steps for the case of the quality assurance rules. The RDF-representation of digital histological slides, which references the medical ontology library, is introduced in Section 4. The last section is dedicated to concrete implementation issues.

Chapter 2

Engineering the Pathology Ontology Library

2.1 Engineering Methodology

Several methodologies have been published in the last decade to predefine the process of construction of a knowledge base or an ontology:

- domain analysis (requirements analysis and knowledge acquisition)
- ontology conceptualization
- implementation of a prototype
- ontology evaluation and refinement
- ontology evolution and maintenance

In our setting we identified the following subtasks, which guided the ontology engineering process:

- analysis of the application domain: we identified the sub-domains of medicine which should be tackled by the ontology: anatomy of the lung, typical diseases, immunohistology, typical routines in pathology diagnostics, the content of pathology reports etc. Useful knowledge sources, potentially relevant for the knowledge base: UMLS, domain knowledge of the experts, medical reports available at the Institute for Pathology Charite.
- requirements analysis: the usage of the ontology for semantic annotation purposes requires additionally that the ontology reflects to a maximal extent the content of the pathology reports. A second requirement for the ontology formalization is related to the text analysis task itself: the knowledge formalization as concepts and relations should be "linguistics-friendly", i.e. the names denominating ontological primitives should be

in a linguistic-predictable form. the usage of the ontology for retrieval purposes requires a formal representation of the content. The usage of Semantic Web technologies was a third pre-defined requirement, since one of the main goals of the project is the evaluation of the implications of using such technologies for medical information systems (Medicine is a often-cited use case for Semantic Web technologies).

- ontology conceptualization: we generated two ontology libraries based on two alternative engineering procedures. The first approach was reuse-oriented, customizing UMLS to the particular application needs. The second approach was completely based on the archive of medical reports.

- ontology implementation: in the first approach the ontology was generated automatically from UMLS, by translating the database-based UMLS data model – identified to be domain-relevant (see below) – to OWL. The second experiment, which is based on knowledge acquisition used the OntoSeed suite, a tool generating statistical reports on text corpora, required a semi-automatical implementation.

- ontology evolution and refinement: we developed a tool for ontology engineering: populating the ontology is realized by the NLP component, which extracts valid ontology concepts from XML-formatted pathology reports. Besides, the component reveals important information about the degree the current (domain) ontology covers the concrete knowledge and terminology formalized by the real users, which are also authors of the pathology reports.

In the following sections we will focus on the generation of the domain ontology and related engineering tasks. As mentioned before we analyzed two conceptualization approaches to model pathology-relevant knowledge: a reuse-oriented approach based on existing ontologies (Section 2.1.1) and a knowledge acquisition approach based on the linguistic analysis of a corpus of pathology reports (Section 2.1.2). For both scenarios support tools have been implemented to aid the ontology design.

2.1.1 Reuse-oriented Ontology Engineering

As input for the medical knowledge base we used UMLS, as the most complex medical thesaurus currently available. Before getting into engineering details we give an overview of UMLS and related ontology libraries: the structure and particularities of these medical ontologies are important for some of the engineering decisions we were confronted with in the project. UMLS as in the current release contains over 1,5 million concepts from over 100 medical libraries and is permanently growing. New sources and current versions of already integrated sources are mapped to the UMLS knowledge format.

UMLS

UMLS defines a common data format for the storage and management of medical information, integrating and mapping standalone libraries. The knowledge is formalized on two levels (Figure 2.2):

- the UMLS Semantic Network: a core ontology containing generic concepts (e.g. "Entity") and core medical concepts (e.g. "Disease", as well as semantical relationships e.g. "location-of", "diagnosed-by"). The network contains approximately 150 concepts and 50 relationships and plays an important role in UMLS, since every concept in the thesaurus is defined as a sub-concept of a Semantic Network concept.

- the UMLS Metathesaurus: a comprehensive semantic network (approximately 1,5 million concepts) formalizing concepts with lexical and synonymic variants (see Figure 2.1). Every concept, term or lexical string is uniquely identified by ids. UMLS also manages information about the origin of every entry and versioning issues. Concepts in Metathesaurus are connected by means of 10 types of relationships (PAR, CHD, SIB, AQ, BQ, RO, RL, RB, RN, SY) A peculiarity of the UMLS data format is the meaning of the "relation attributes" used for some of the Metathesaurus relations. The relation attribute references a semantic relation from the Semantic Network, but its exact meaning in the context of the current concept pair depends on the associated Metathesaurus relation. E.g. the combination "associated_with" (a relation from the Semantic Network) and "PAR/parent" (a relation from the Metathesaurus) means a *direct* relationship between the concepts, while the same attribute together with the Metathesaurus relation "RB/broader" implies an indirect relationship between the concepts (i.e. something like a path of length greater than 1 between the concepts). The absence of a relation attribute reduces the Metathesaurus relations to their original meaning, e.g. a relation "CHD/child" with no attribute is interpreted as "subClassOf".

We now turn to the usage of UMSL in generating the ontology of lung pathology. Due to the complexity of the thesaurus and the limitations of current Semantic Web tools we need to customize it w.r.t. to two important axes:

- 1) the *identification of relevant libraries and concepts:* Which UMLS libraries and parts of libraries describe medical sub-domains which are relevant in lung pathology? and

- 2) their *adaptation* to the particularities of language and vocabulary of the case report archive.

Identifying application-relevant knowledge in UMLS

In a "pre-selection" phase domain experts reduced the huge amount of medical information from UMLS to the domain "lung pathology". They identified

<table>
<tr><th>Concept (CUI)</th><th>Term (LUI)</th><th>String (SUI)</th></tr>
<tr><td rowspan="4">C0004238
Atrial Fibrillation
(preferred)
Atrial Fibrillations
Auricular Fibrillation
Auricular Fibrillations</td><td rowspan="2">L0004238
Atrial Fibrillation
(preferred)
Atrial Fibrillations</td><td>S0016668
Atrial Fibrillation
(preferred)</td></tr>
<tr><td>S0016669
Atrial Fibrillations</td></tr>
<tr><td rowspan="2">L0004327
(synonym)
Auricular Fibrillation
Auricular Fibrillations</td><td>S0016899
Auricular Fibrillation
(preferred)</td></tr>
<tr><td>S0016900
(plural variant)
Auricular Fibrillations</td></tr>
</table>

Figure 2.1: Data Model for Concepts in UMLS

potentially relevant UMLS libraries. The large number of partially overlapping libraries and the complexity of their interdepedencies made this process time-consuming and error-prone, so that the final goal of the "pre-selection" phase was to exclude libraries, which are definitively irrelevant to our application domain.

Approximately 50 percent of the UMLS libraries were selected as *possibly* relevant for lung pathology, containing more than 500000 concepts (see Appendix **??**). Managing an ontology of such dimensions with Semantic Web technologies is related to still unsolved issues w.r.t. to scalability and performance of the system. In the second step we used the case reports archive to identify concepts, which actually occur in medical reports. These concepts are really used by pathologists when putting down their observations and therefore will also occur as search parameters. We compared the vocabulary of the reports archive to the content of the preselected UMLS libraries by means of a retrieval engine. The result of this task was a list of 10 UMLS libraries, still containing approximately 350,000 different concepts:

- Digital Anatomist
- MeSH
- MeSH German
- SNOMED 1982
- SNOMED 1998 International
- UMLS Semantic Network
- ICD10, ICD10 AM Engl

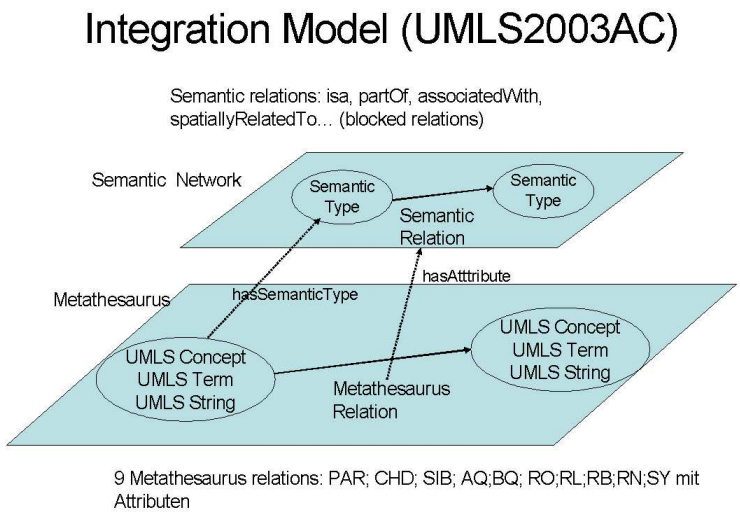

Figure 2.2: Integration Model in UMLS

- ICD10 German
- ICD-9-CM
- Read Codes RCD

The size of the concept set can be explained if we consider the fact that the UMLS knowledge is concentrated in few major libraries (e.g. MeSH, SNOMED98, Digital Anatomist), which cover important parts of the complete thesaurus and therefore contain the most of the concepts in our lexicon. To differentiate among the concepts within the resulted 10 libraries, pathology experts selected 4 central concepts in lung anatomy (i.e. "lung", "pleura", "trachea" and "bronchia") and extracted similar or related concepts from UMLS libraries. They considered the list of all distinct concepts related through a relation of any kind to the 4 initial concepts (see the UMLS data model above). The result was a set of approximately 1000 concepts describing the anatomy of the lung and lung diseases and served as initial input for the domain ontology.

Adapting the ontology to the application domain

The linguistic analysis of the patient report corpus evidenced the content-related limitations of UMLS w.r.t. the concrete vocabulary of the report archive. We modelled additional pathology-specific concepts, like the components and typical content of a medical report (see Figure ...) , and integrate them in the available ontology library. Besides content-related adaptation needs, the anal-

ysis of the generated ontology outlined several "syntactical" issues for further adaptations:

- concept names in UMLS: concepts like "ARF-smaller-then-2", "RESECTION OF LUNG WITH RECONSTRUCTION OF CHEST WALL WITHOUT PROSTHESIS", "Unspecified injury of lung with open wound into thorax" are unlikely to be relevant to the retrieval of pathology reports. Besides, they should be modelled as concepts with corresponding properties and not directly as a single concept, whose name denotes its meaning.

- the absence of concept names in German language: due to the predominance of English in denominating UMLS concepts and the predominance of German terms in the pathology report archive in our application setting one needs to translate the English terms in order to achieve an efficient retrieval.

The comparison of the vocabulary of the medical reports archive with the generated ontology also emphasized the need to extend the knowledge base with non-medical content. Especially part-whole and spatial relationships are often encountered in medical findings and are therefore included to the ontology library. Medical reports frequently contain ambiguous terms to describe the results of the examinations (e.g. terms like "high-grade", "low-grade","slightly polymorphic"), which play an important role for the overall interpretation of the reports. The representation of such terms is still subject of future work.

OWL Representation

After identifying the relevant knowledge sources and the list of concepts which can be used as input for our application, we translated the UMLS data model to the OWL model and transformed the relevant data from one format to another. We implemented a Java-based module, which reads the UMLS data from a relational database and generates the corresponding OWL constructs using Jena2 (see Section 5.1. The resulting ontologies are published server-side and can be accessed by all components in the system.

The list of application-relevant concepts is part of the Metathesaurus and therefore each of the concepts is subsumed by semantic types. First we translated the UMLS Semantic Network to OWL and created a taxonomy of semantic types as classes and a taxonomy of semantic relations as properties. A second ontology contains the UMLS concepts; every UMLS concept is transformed in an OWL class. The Metathesarus relations "parent" and "child" are formalized as OWL "subClassOf" constraints. The "narrower" and "broader" relations , which define indirect subsumption relations, are formalized as "ancestor" and "descendant" in the OWL ontology. These relations could also be ignored, since their meaning can be inferred from the ontology using a reasoner. Due to the fuzzy definition of the rest of the Metathesaurus relations, we merged them to a single "related_to" property. The connection between relations and relation attributes is also considered in the ontology. Since the relation attribute points to

the semantics of a relationship between two concepts, we used this information if available. We considered the Metathesarus relations only for the case where a relation attribute was missing. We store for every concept the list of alternative names together with language specifications as rdfs:label. After translating the UMLS data to OWL we checked the ontologies for consistency and analyzed the inferred classification hierarchy, which pointed out few differences compared to the original UMLS hierarchy.

2.1.2 Text-oriented Ontology Engineering

Due to the difficulties related to the usage and customization of UMLS and its component libraries and the poor coverage of the resulted ontology w.r.t. the medical reports to be indexed by it (see Section 2.3, we analyzed a second approach to generate the ontology for lung pathology. In comparison to the first approach we used only text documents, which were analyzed linguistically, and did not resort to any existing knowledge source. For the generation of the target ontology a corpus of 400 pathology reports from the Institute of Pathology Charite were provided as input to generate a suite of statistical reports, which were subsequently used as documentation for the ontology engineers (a computer scientist from FU Berlin and two pathologists from the Charite Hospital). More specifically we used the following 5 information sources reflecting the content of the pathology documents:

- a list of nouns sorted by their domain relevance
- a hierarchically organized list of nouns grouped by their prefixes and a second grouping by common suffixes
- a list of adjectives (modifiers) with the nouns they modify
- the same list of modifiers, indexed by modified nouns.

Examples of such lists are illustrated in Appendix...

Pre-processing the content of pathology reports in this way has proved to be useful for the ontology design. Though the process can not be performed automatically, the statistical listings described above offer valuable information which can ease modelling decisions as follows:

- selecting relevant concepts: the weighting function used to sort the nouns in the text collection allowed us to reduce the time necessary for the evaluation of the entire vocabulary, since 80% of the relevant concepts are in the first half of the sorted list.
- prefixes give hints about potential parts and properties of a given concept, while its suffixes can often be interpreted as subclasses.
- the modifier lists offer information about further properties of the selected concepts. The ontology engineer compares the distribution of the modifiers along collection nouns and the distribution of the nouns for each adjective.

Adjectives which are predominantly used in connection with a certain noun (or group of nouns) should be defined for the correspondent concept(s). In the same time the usage of the modifiers with several types of concepts may indicate a generic property.

We employed this approach to generate a text-based ontology for lung pathology. The relevant concepts are still aligned to the UMLS Semantic Network; the integration of such a pre-defined core level had proven to be advantageous since it offers an initial matrix for the conceptualization and general purpose semantic relationships, which were not considered directly during the text processing phase. The structure of pathology reports, as well as immunohistology guidelines – encoded as informal textual descriptions – were formalized manually as in the first engineering approach.

The OWL implementation of the ontology, given the set of domain-relevant ontological primitives, was performed manually using the well-known ontology management tool Proteg. A first step in this direction was realized automatically: the generation of basic OWL classes for the set of relevant concepts. The taxonomy and the definition of complex concepts – usually denominated by compound nouns (German) or noun phrases (English) – were performed manually by means of the mentioned editing tool.

A first evaluation of the ontologies was performed by comparing their content and coverage with a second corpus of pathology reports with a similar size (370 reports). The results were encouraging: only 5% of the relevant concepts belonging to the second corpus was found not to be covered by the ontologies. This outcome suggests that the ontologies generated on the basis of the first corpus are sufficiently representative for the application domain and that their evolution at least along to this axis is not problematic. By comparison to the UMLS-based ontologies, we found out that the corpus-based ontology covered 10 times as many terms encountered in the text documents. This result motivated us to use the latter ones in the implementation of our ontology-based retrieval system.

Since both conceptualization approaches rely on the same requirements analysis, the outcomes differ only in details and granularity of the represented content, but cover the same application domain lung pathology. In the following we describe the generated ontology library, which is used as input for further processing tasks in the retrieval system.

2.2 The Ontology Library

The result of the engineering process is a library of ontologies describing both application-independent and application-specific knowledge. For reusability purposes these two levels are formalized in separate modules:

1. UMLS Semantic Network (umlssn.owl): implementation of the UMLS Semantic Network in OWL. In the text-oriented approach we simplified the

semantic network to our application needs. Therefore in this case the ontology is a proper sub-model of the original UMLS correspondent.

2. ontology of lung anatomy and lung diseases (swpatho1.owl: UMLS-based, swpatho2.owl: corpus-based)

3. ontology of pathology reports (befundbericht.owl): contains concepts related to specific pathology procedures, such as types of reports, report components (microscopy, macroscopy – see the Report on AP1 for a detailed description), typical pathology compounds, such as biopsies, exudates etc.

4. ontology of immunohistology (immunohistology.owl): contains concepts such as markers and stains used as diagnostic aid.

5. upper level ontology (swpatho_upperlevel.owl): simple ontology modelling part-whole relationships and spatial relationships among geometrical shapes.

Figure 2.3 illustrates the dependencies within the ontology libraries (an edge from ontology A to ontology B means that the former imports the latter).

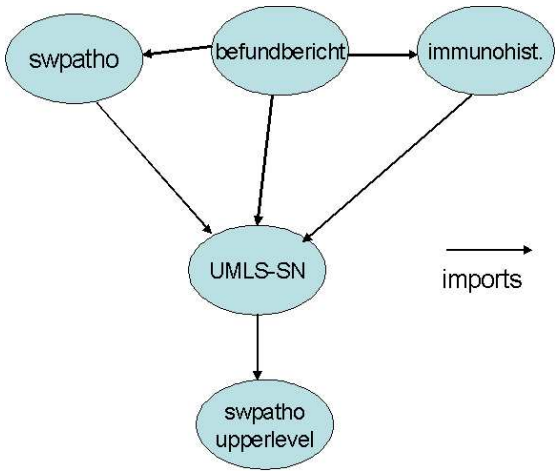

Figure 2.3: SWPATHO Ontology Library

The ontologies are public and can be accessed at:

`http://nbi.inf.fu-berlin.de/research/swpatho/owldata/swpatho1/<ontologyName>.owl`

for the UMLS-based experiment and

`http://nbi.inf.fu-berlin.de/research/swpatho/owldata/<ontologyName>.owl`

for the second experiment respectively.

2.3 Lessons learned

Though containing significant amounts of valuable knowledge UMLS and related medical knowledge sources can not be easily integrated to Semantic Web applications. Our experiences in this project showed that at least for our application setting knowledge acquisition is much more suitable as method to generate input for application ontologies. The second approach created an ontology which can be evaluated by domain experts and does not require laborious corrections or extensions. From a costs point of view the second engineering experiment took 25% of the time invested in the UMLS-based approach.

The UMLS contains several problematic modelling decisions which have been often described in research projects aiming to integrate it in knowledge-based applications. Still, a comprehensive analysis of the quality of UMLS in such a setting or especially for Semantic Web applications has not delivered an optimal solution to cope with this problem. A possible start point could be the Semantic Network, since every Metathesaurus concept is related to it. Besides, the Semantic Network is supposed to be independent of a particular area in medicine. At this point it is not clear how important such issues are for the quality of our retrieval system, but we intend to extend the Semantic Network with a more detailed and coherent upper level ontology. Besides, there is a need for powerful tools which enable a flexible customization of the comprehensive thesaurus beyond language and library restrictions. Such a tool would have been an important incentive to reuse UMLS in our setting.

On the other side since the corpus-based approach, as in its actual implementation, does not offer any support for semantical relationships, this kind of knowledge has to formalized explicitly and manually by domain experts. Parts of this knowledge would surely have been already encoded in UMLS. Reusing UMLS means also increasing interoperability between our system and the wide range of medical systems using UMLS for various purposes. However we think that for interoperability issues a integration of our ontology library to well-known ontologies like SNOMED and UMLS is more advantageous and cost-saving than a pure UMLS reuse. In the latter case we are confronted with the problems related to the customization of UMLS and the difficulties in evaluating the customized sub-ontology, while the former case offers the benefit of having a high-quality ontology, which is balanced between application-specificity and reusability.

Representing medical knowledge using Description Logics is not a trivial task. Although translating the UMLS data format to OWL was a straightforward procedure, the expressivity limitations of the language become clear after a detailed analysis of the semantics of the medical knowledge. Reasoning beyond subsumption hierarchies and an extended support for concrete domains are very important for an efficient semantic retrieval system.

Chapter 3

Engineering the Rule-based Knowledge Base

This section describes our work in the tasks 2.5 and 2.6 of the second work package. These two tasks aimed at providing a technological base for the formalization and processing of Semantic Web enabled domain rules, which were intended to extend the ontology-based knowledge base with additional facts.

The rule set is an extension of the already generated OWL-based pathology ontology library. Their meaning is twofold: first we formalize by means of rules pathology-specific knowledge which can not be expressed in OWL-DL due to expressivity limitations; Second we formalize quality management guidelines using rules, because this type of formalization (e.g. if-then rules) is perceived as more intuitive by the domain experts.

However the usage of ontology- and rule-based knowledge in a common framework requires the development of a reasoning engine which is able to deal with both. This issue is not solved completely by the current Semantic Web research efforts (see Section 3.5.

3.1 Engineering Methodology

Basically for the realization of the rule based knowledge base we used the same engineering methodology as for the ontology library:

- domain analysis (requirements analysis and knowledge acquisition)
- rules conceptualization
- implementation of a prototype rule base
- evaluation and refinement
- evolution and maintenance

In our setting we identified the following subtasks, which guided the rules engineering process:

- domain and requirements analysis: we identified the scope of the target rule base: parts of the pathology domain which can not be formalized using OWL-DL and quality assurance guidelines required by the quality management component. Useful knowledge sources emerged from the ontology engineering task in AP2.1. A second input was provided by the domain experts from the Institute of Pathology, Charité, who put together a list of the most significant quality criteria used in their institute for the currently manually realized quality assurance procedures.
- rules conceptualization: rules reference concepts from the generated ontology library, put new constraints on the ontological knowledge and formalize quality criteria respectively.
- rules implementation: rules are implemented manually using SWRL and RuleML. Due to the reasoning problems related to the combination of OWL and rule-based knowledge bases, a special attention has been paid in restricting the rule form to subsets of FOL which have been identified as advantageous w.r.t. complexity and reasoning capabilities by recent research results in the Semantic Web community (e.g. DL-safe SWRL rules).
- evolution and refinement: will be performed after the implementation of the quality assurance component due to month 24.

3.2 OWL2Jess

In order to enable reasoning over ontology- and rules-based knowledge bases, we realized OWL2Jess, a hybrid reasoning framework (Figure-3.1), which can be used to fill the gap between OWL and Jess[1], the Java Expert System Shell. Jess is a Java-based rule engine and scripting environment. While domain knowledge is still modelled with OWL, using common ontology editors, we transform this OWL formalism to Jess facts using XSL transformations on the XML-syntax of OWL and represent additional rules in Jess. In addition to our predefined entailment rules on the basis of RDF semantics and OWL (RDF-compatible) Model-Theoretic Semantics, we run the Jess rule engine to implement the reasoning services.

Note that the predefined rules are used to check the consistency, to compute the taxonomic classification, the characteristics of RDF/OWL vocabulary etc. The inferred Jess assertions are helpful for the ontology engineer to evaluate and refine the original OWL ontology.

According different expressivity levels, OWL2Jess actually could be either reduced to pure RDF2Jess, or extended to newly SWRL2Jess where SWRL extends the set of OWL axioms to include Horn-like rules. By converting OWL

[1] http://herzberg.ca.sandia.gov/jess/

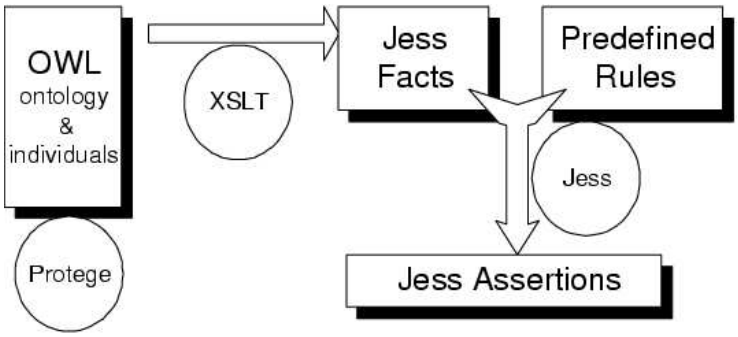

Figure 3.1: OWL2Jess Model

syntactically and semantically to Jess the tool enables the usage of the Jess reasoning engine over OWL ontologies, which might be more efficient than DL reasoners for particular tasks. Besides, when extending an OWL-formalized knowledge base with rules, one needs a common reasoning engine for the corresponding heterogeneous data base, while preserving the advantages of using OWL for particular modelling tasks.

3.2.1 Transforming OWL to Jess

Converting OWL knowledge bases to Jess in realized in four steps.

1). The first step is to build the ontology. An ontology editor like Protege[2] provides an OWL Plug-in to support the development of OWL ontologies. Organizing knowledge in terms of classes, properties, restrictions and individuals has been proven to be well accepted by domain experts and software developers, since this paradigm is very similar to object-oriented modelling or UML. Besides in the last decades various ontologies for almost every application domain have been formalized in RDF(S) and OWL and can be therefore re-used to be extended with rules if necessary.

2). The second step is to transform the XML syntax of OWL into the Jess syntax by means of XSLT. Starting from the root, recurrent processes of ABox-class and ABox-property are called via a set of named templates. The output file consists of Jess facts. If the semantics of the underlying ontology languages is already specified as Jess rules, specific keyword matching is unnecessary in the OWL2Jess XSL transformation.

```
<xsl:template name="ABox-property" >
```

[2]http://protege.stanford.edu/

```
(assert (triple
 (predicate "<xsl:value-of select="concat(ns-uri(.),name(.))"/>")
 (subject   "<xsl:call-template name="get-father-ID"/>")
 (object    "<xsl:call-template name="get-child-ID"/>"))
)<xsl:for-each select="*[position()=1]">
  <xsl:call-template name="ABox-class"/>
 </xsl:for-each></xsl:template>
```

3). The third step is to combine the Jess files, including the result of XSLT, and our predefined RDF/OWL entailment rules. Furthermore, external Jess-style queries and rulers can also be appended, such as the composition of properties (like "hasUncle(x,z) ← hasFather(x,y), hasBrother(y,z)"). Such rules could also be represented using the SWRL rule language and the SWRL2Jess transformation tool.

4). The fourth step is to run the Jess rule engine. Among our predefined rules, we mention consistency checking, classification and characteristics. Output results with error messages indicate invalid or illegal issues in the incoming OWL ontology. Caution messages are used to signal whether the engine has discovered an individual belonging to a certain class, moreover the machine would randomly deal with the uncertainty like $\exists$ or $\vee$, based on the currently given knowledge base.

3.2.2 Variants of OWL2Jess

Considering OWL (RDF-compatible) Model-Theoretic Semantics is based on RDF Semantics, our Jess rule file "owlmt.clp" includes a separate "rdfmt.clp", resulting that a pure RDF document would also be applied using a simpler RDF2Jess model, which will avoid unnecessary, expensive computations on the more complex OWL semantics.

Another possible alternative to OWL2Jess is its extension to support SWRL-enhanced OWL ontologies. Consequently, by another "SWRL2Jess.xsl" stylesheet, the translation from the SWRL syntax to Jess syntax could be easily accomplished. However, unlike the above OWL2Jess XSL transformation, we need additional keyword matching template to distinguish among different types of SWRL rules and their components (see the XSLT fragment below). The output consists of Jess rules, which will be handled in the same way as other entailment rules sharing the common ontology knowledge.

```
 <xsl:template match="ruleml:Imp">
(defrule imp <xsl:apply-templates select="ruleml:body" />
         =>  <xsl:apply-templates select="ruleml:head" />
)</xsl:template>

 <xsl:template match="swrl:ClassAtom">
(triple
 (predicate "&rdf;#type")
```

```
 (object    <xsl:apply-templates select="swrl:argument1" />)
 (subject   <xsl:apply-templates select="swrl:classPredicate" />)
)</xsl:template>
```

3.2.3 Some Issues about the OWL to Jess Transformation

As mentioned above, our proposal suggests to build ontologies in OWL and transform the OWL knowledge base to Jess for particular reasoning purposes. However, the semantics of OWL currently adopts an open world assumption(OWA), while all rule-based languages including Jess are based on a close world assumption(CWA).

In OWA, everything which was not specified explicitly is unknown to the reasoning service. For example, an owl:Class is defined as $C = \forall P.D$, but we can not conclude $u \in C$ even if we have currently found out "for any given $v, P(u, v)$ there is $v \in D$". The reason is there are more unknown t, maybe $P(u, t)$ but $t \notin D$. This fact can not be derived automatically due to the open world assumption. However, in practice, we indeed need such real-time conclusions, especially when we want to know whether there is something wrong or something missing about our existing ontology.

Consequently, our intention is merely to check the given OWL ontology, to remind the errors or cautions, and to suggest the modifications, rather than attempt to modify it. According our error or caution messages, the author can revise the ontology manually. Furthermore, the inferred Jess assertions are helpful for the author to recognize all characteristics of the ontology, such as that an individual currently belongs to an OWL restriction or an OWL boolean combination, even if this assertion is missing from the original ontology.

Another important issue is related to "if-and-only-if"(iff) conditions. In RDF Semantics, there are extensional semantic conditions (i.e., iff conditions) for rdfs:subClassOf, rdfs:subPropertyOf, rdfs:domain, and rdfs:range, which are so strong that some consequences inferred are useless in practice. For instance, the domain and range of every property are extended to the largest one, namely rdfs:Resource, resulting in confusions with original definitions. A similar situation appears for the OWL (RDF compatible) Semantics. We ignore these extensions here, however they are easy to be included if they are required.

3.3 Entailment Rules

According to the RDF Semantics and the OWL (RDF-compatible) Model-Theoretic Semantics, we implement the entailment rules in Jess one by one. In the following we present the most important ones with a focus on owl:Restrictions and boolean expressions, which are viewed as expressive restrictions for rule-based languages.

Some works-around are helpful to cope with the semantic discrepancy between OWL and rule-based languages: error messages indicating some illegal

or invalid issues in the ontology or caution messages pointing out potentially missing ontology statements.

3.3.1 RDF Semantics

RDF(S) axiomatic triples are transformed to facts. The following is a simple Jess fact for "rdf:type rdf:type rdf:Property".

```
(deffacts RDF_axiomatic_triples
  (triple (predicate "rdf:type")
          (subject   "rdf:type")
          (object    "rdf:Property")))
```

It is unnecessary to assert the triples mentioned as RDFS-valid, such as "rdfs:Class rdf:type rdfs:Class", since all these could be inferred by other existing rules.

RDF(S) semantic conditions are transformed to rules, some of which are presented as follows. We ignore the extensional semantic conditions, i.e., if-and-only-if conditions for rdfs:subClassOf, rdfs:subPropertyOf, rdfs:domain, and rdfs:range, which are so strong that some consequences inferred are useless in practice. For instance, the domain and range of every property are extended to the largest one, namely rdfs:Resource, which would confuse with our intention. Of course, the same situations also appear in OWL semantics, however they are easy to be included if they are required.

rdf:type

It is stated as a basic RDFS semantic condition that, x is in ICEXT(y) if and only if $< x, y >$ is in IEXT(I(rdf:type)). Thus, we translate the inclusion relation into a rule as below:

```
(defrule RDFS_semantic_conditions_type
  (triple (predicate "rdf:type") (subject ?s) (object ?o))
  =>
  (assert (triple (predicate "rdf:type")
                  (subject   ?o)
                  (object    "rdfs:Class"))))
```

rdfs:domain

It is stated in RDF Semantics that, if $< x, y >$ is in IEXT(I(rdfs:domain)) and $< u, v >$ is in IEXT(x) then u is in ICEXT(y). The corresponding rule is as follows:

```
(defrule RDFS_semantic_conditions_range
  (triple (predicate "rdfs:domain") (subject ?x) (object ?y))
  (triple (predicate ?x) (subject ?u) (object ?v))
  =>
  (assert(triple (predicate "rdf:type") (subject ?u) (object ?y))))
```

rdfs:range

It is stated in RDF Semantics that, if $< x, y >$ is in IEXT(I(rdfs:range)) and $< u, v >$ is in IEXT(x) then v is in ICEXT(y). The corresponding rule is as follows:

```
(defrule RDFS_semantic_conditions_range
  (triple (predicate "rdfs:range") (subject ?x) (object ?y))
  (triple (predicate ?x) (subject ?u) (object ?v))
  =>
  (assert(triple (predicate "rdf:type") (subject ?v) (object ?y))))
```

rdfs:subPropertyOf

First rdfs:subPropertyOf is transitive and reflexive.

```
(defrule RDFS_semantic_conditions_subPropertyOf_transitive
  (triple (predicate "rdfs:subPropertyOf") (subject ?x) (object ?y))
  (triple (predicate "rdfs:subPropertyOf") (subject ?y) (object ?z))
  =>
  (assert (triple (predicate "rdfs:subPropertyOf")
                  (subject ?x) (object ?z))))

(defrule RDFS_semantic_conditions_subPropertyOf_reflexive
  (triple
   (predicate "rdf:type")
   (subject   ?p)
   (object    "rdf:Property"))
   =>
  (assert (triple (predicate "rdfs:subPropertyOf")
                  (subject ?p) (object ?p))))
```

Moreover it has the following characteristics:

```
(defrule RDFS_semantic_conditions_subPropertyOf
   (triple (predicate "rdfs:subPropertyOf") (subject ?x) (object ?y))
   (triple (predicate ?x) (subject ?a) (object ?b))
   =>
   (assert (triple (predicate ?y) (subject ?a) (object ?b))))
```

rdfs:subClassOf

rdfs:subClassOf has the same properties as rdfs:subPropertyOf.

```
(defrule RDFS_semantic_conditions_subClassOf_transitive
  (triple (predicate "rdfs:subClassOf") (subject ?x) (object ?y))
  (triple (predicate "rdfs:subClassOf") (subject ?y) (object ?z))
  =>
```

```
  (assert (triple (predicate "rdfs:subClassOf")
                  (subject ?x) (object ?z))))

(defrule RDFS_semantic_conditions_subClassOf_reflexive
  (triple
   (predicate "rdf:type")
   (subject   ?c)
   (object    "rdfs:Class"))
   =>
  (assert (triple (predicate "rdfs:subClassOf")
                  (subject ?c) (object ?c))))

(defrule RDFS_semantic_conditions_subClassOf
   (triple (predicate "rdfs:subClassOf") (subject ?x) (object ?y))
   (triple (predicate "rdf:type") (subject ?o) (object ?x))
   =>
   (assert (triple (predicate "rdf:type") (subject ?o) (object ?y))))
```

Moreover, it was pointed out that, any class is a subclass of the largest class, namely rdfs:Resource.

```
(defrule RDFS_semantic_conditions_subClassOf_Resource
  (triple (predicate "rdf:type")
          (subject    ?x)
          (object    "rdfs:Class"))
  =>
  (assert (triple (predicate "rdfs:subClassOf")
                  (subject   ?x)
                  (object    "rdfs:Resource"))))
```

3.3.2 OWL Semantics

The implementation of the OWL semantics are more challenging, since OWL is an extension of RDFS to provide restrictions on how properties behave in a local class scope. We define additional facts to represent typical OWL primitives and their relationship to RDFS such as owl:Class is subclass of rdfs:Class. A set of rules are defined to represent characteristics of OWL classes, datatypes, properties and restrictions. In the remaining of this section we restrict to OWL restrictions and boolean expressions, which are in our opinion the most relevant for the OWL to Jess conversion.

OR condition

We firstly encounter the "or" condition in the head of a rule, since subject-values for owl:AnnotationProperty are owl:Thing or rdfs:Literal. Subsequently $B \vee C \leftarrow A$ equals to $\neg A \vee B \vee C$ equals to $C \leftarrow A \wedge \neg B$, while the first one cannot be expressed in Jess and the last one can. However, when we represent

it as follows, by default the subject-values are owl:Thing if there is no definition in advance. Usually such precise definitions are provided by the ontology author in advance or might be suggested by ontology editors.

```
(defrule OWL_characteristics_AnnotationProperty_object
  (triple (predicate "rdf:type")
          (subject   ?e)
          (object    "owl:AnnotationProperty"))
  (triple (predicate ?e) (subject ?x) (object ?y))
  (not (triple (predicate "rdf:type")
               (subject   ?y)
               (object    "rdfs:Literal")))
  =>
  (assert (triple (predicate "rdf:type")
                  (subject   ?y)
                  (object    "owl:Thing")))
  (printout t "Caution!" ?y " now is in owl:Thing" crlf))
```

EQUIVALENT condition

owl:equivalentClasses and owl:equivalentProperties are stated to be subclasses or subproperties of each other.

```
(defrule OWL_characteristics_equivalentProperty_relationship
  (triple (predicate "owl:equivalentProperty") (subject ?x) (object ?y))
  (test (neq 0 (str-compare ?x ?y)))
  =>
  (assert (triple (predicate "owl:subPropertyOf")
                  (subject ?x) (object ?y)))
  (assert (triple (predicate "owl:subPropertyOf")
                  (subject ?y) (object ?x))))
```

In order to deal with the owl:disjointWith constraint, we make use of owl:differentFrom to state the individuals of the two classes are different from each other.

```
(defrule OWL_characteristics_disjointWith_relationship
  (triple (predicate "owl:disjointWith") (subject ?x) (object ?y))
  (triple (predicate "rdf:type") (subject ?o1) (object ?x))
  (triple (predicate "rdf:type") (subject ?o2) (object ?y))
  =>
  (assert (triple (predicate "owl:differentFrom")
                  (subject ?o1) (object ?o2))))
```

The owl:differentFrom and owl:sameAs constructs can not be translated directly to Jess. Checking $x \neq y$ or $x = y$ is easy, but no further statement could be provided about its meaning. Consequently, an error message is generated to signal this issue to the user.

```
(defrule OWL_characteristics_differentFrom
  (triple (predicate "owl:differrentFrom") (subject ?x) (object ?y))
  (test (eq 0 (str-compare ?x ?y)))
  =>
  (printout t "Error!" ?x " is not different from " ?y crlf))

(defrule OWL_characteristics_sameAs
  (triple (predicate "owl:sameAs") (subject ?x) (object ?y))
  (test (neq 0 (str-compare ?x ?y)))
  =>
  (printout t "Error!" ?x " is not same as " ?y crlf))
```

owl:complementOf

owl:complementOf is a subproperty of owl:disjointWith. It should be noticed that owl:complementOf has a very strong semantics, resulting in any individual must be in a class or in its complement, once the individual has been stated in the universe. We state the meaning of owl:complementOf by this rule:

```
(defrule OWL_complementOf
  (triple (predicate "owl:complementOf") (subject ?x) (object ?y))
  (triple (predicate "rdf:type") (subject ?u) (object "owl:Thing"))
  (not (triple (predicate "rdf:type") (subject ?u) (object ?y)))
  =>
  (assert (triple (predicate "rdf:type") (subject ?u) (object ?x)))
  (printout t "Caution!" ?u " now is in " ?x crlf))
```

owl:intersectionOf

In OWL, the subject-value of owl:intersectionOf is a sequence of rdf:first, rdf:rest constructs. However, via XSLT, we directly catch a owl:Class as the subject-value, hence we skip the verbose syntax of rdf:List. Moreover, in Set Theory, set equation $A = B$ means $A \subseteq B$ and $A \supseteq B$, so we decompose the set equation of owl:intersectionOf into two rules "subset" and "supset". Surely owl:unionOf and owl:oneOf can be treated in a similar way.

Suppose $\langle x, y\rangle \in EXT_I(S_I(\text{owl:intersectionOf}))$ and y is a sequence of $y_1, \cdots, y_n$, the subset relation is easy to state, because $CEXT_I(x) \subseteq CEXT_I(y_1) \cap \cdots \cap CEXT_I(y_n) \subseteq CEXT_I(y_i)$. That is, for any $u \in CEXT_I(x)$, we have $u \in CEXT_I(y_i), 1 \leq i \leq n$, as shown below.

```
(defrule OWL_intersectionOf_subset
  (triple (predicate "owl:intersectionOf") (subject ?x) (object ?y))
  (triple (predicate "rdf:type") (subject ?u) (object ?x))
  =>
  (assert (triple (predicate "rdf:type") (subject ?u) (object ?y))))
```

Furthermore, the supset relation is also permitted in Jess. The troublesome issue is to check out the individual u who belongs to all subclasses, i.e., $\forall y_i, u \in CEXT_I(y_i)$. Here, we make use of an implication in the body of a rule, namely $C \leftarrow (B \leftarrow A)$, equals to $C \leftarrow (\neg A \vee B)$. However, it is better to write as $\neg(A \wedge \neg B)$ in Jess, for the negation of Jess is the failure of matching, and what we want to know is the result of the matching of B rather A.

```
(defrule OWL_intersectionOf_supset
  (triple (predicate "owl:intersectionOf") (subject ?x) (object ?y))
  (triple (predicate "rdf:type") (subject ?u) (object ?y))
  (not (and (triple (predicate "owl:intersectionOf")(subject ?x)(object ?v))
            (not (triple (predicate "rdf:type")(subject ?u)(object ?v)))))
  =>
  (assert (triple (predicate "rdf:type") (subject ?u) (object ?x))))
```

3.3.3 owl:unionOf

If $\langle x, y\rangle \in EXT_I(S_I(\text{owl:unionOf}))$ and y is a sequence of $y_1, \cdots, y_n$, we can easily state the supset relation, because $CEXT_I(x) \supseteq CEXT_I(y_1) \cup \cdots \cup CEXT_I(y_n) \supseteq CEXT_I(y_i)$.

```
(defrule OWL_unionOf_supset
  (triple (predicate "owl:unionOf") (subject ?x) (object ?y))
  (triple (predicate "rdf:type") (subject ?u) (object ?y))
  =>
  (assert (triple (predicate "rdfs:type") (subject ?u) (object ?x))))
```

However, the subset relation is troublesome due to the uncertainty. Given an individual $u \in CEXT_I(x)$, if we check out $\forall y_i, u \notin CEXT_I(y_i)$, then it indicates something missing about the subclasses of the union x in the existing ontology, else $u \in CEXT_I(y_i)$ has been satisfiable. The engine would randomly assign u to one y_i after generating a caution message, which suggests the ontology author to assert a new individual belonging to x and certain y_i in the original ontology.

```
(defrule OWL_unionOf_subset
  (triple (predicate "owl:unionOf") (subject ?x) (object ?y))
  (triple (predicate "rdf:type") (subject ?u) (object ?x))
  (not (and (triple (predicate "owl:unionOf")(subject ?x)(object ?v))
            (triple (predicate "rdf:type")(subject ?u)(object ?v))))
  =>
  (printout t "Caution!" ?u " now is in " ?y crlf)
  (assert (triple (predicate "rdf:type") (subject ?u) (object ?y))))
```

3.3.4 owl:oneOf

To some extend, owl:oneOf is similar to owl:unionOf, for $\{y_1, \cdots, y_n\} = \{y_1\} \cup \cdots \cup \{y_n\}$. Consequently in the subset relation, rdf:type is changed into owl:sameAs.

```
(defrule OWL_oneOf_subset
  (triple (predicate "owl:oneOf") (subject ?x) (object ?y))
  (triple (predicate "rdf:type") (subject ?u) (object ?x))
  (not(and(triple (predicate "owl:oneOf")(subject ?x)(object ?v))
        (or(test(eq 0 (str-compare ?u ?v)))
           (triple(predicate "owl:sameAs")(subject ?u)(object ?v)))))
  =>
  (printout t "Caution!" ?u " now is same as " ?y crlf)
  (assert (triple (predicate "owl:sameAs") (subject ?u) (object ?y))))
```

The supset relation is also more simple than in the previous cases:

```
(defrule OWL_oneOf_supset
  (triple (predicate "owl:oneOf") (subject ?x) (object ?y))
  =>
  (assert (triple (predicate "rdf:type") (subject ?y) (object ?x))))
```

owl:allValuesFrom

Suppose $\langle x, y\rangle \in EXT_I(S_I(\text{owl:allValuesFrom}))$ and $\langle x, p\rangle \in EXT_I(S_I(\text{owl:onProperty}))$, the subset relation is $CEXT_I(x) \subseteq \{u|\langle u, v\rangle \in EXT_I(p)$ implies $v \in CEXT_I(y)\}$, which can be easily translated into a Jess rule as below:

```
(defrule OWL_allValuesFrom_subset
  (triple (predicate "owl:allValuesFrom") (subject ?x) (object ?y))
  (triple (predicate "owl:onProperty") (subject ?x) (object ?p))
  (triple (predicate "rdf:type") (subject ?u) (object ?x))
  (triple (predicate ?p) (subject ?u) (object ?v))
  =>
  (assert (triple (predicate "rdf:type") (subject ?v) (object ?y))))
```

The supset relation contains an implication in the body of a rule, in which $B \leftarrow A$ is transformed as $\neg(A \wedge \neg B)$, where $A = \langle u, v\rangle \in EXT_I(p), B = v \in CEXT_I(y)$.

```
(defrule OWL_allValuesFrom_supset
  (triple (predicate "owl:allValuesFrom") (subject ?x) (object ?y))
  (triple (predicate "owl:onProperty") (subject ?x) (object ?p))
  (triple (predicate ?p) (subject ?u) (object ?v))
  (not (and (triple (predicate ?p) (subject ?u) (object ?o))
            (not (triple (predicate "rdf:type") (subject ?o) (object ?y)))))
  =>
  (assert (triple (predicate "rdf:type") (subject ?u) (object ?x))))
```

owl:someValuesFrom

Suppose $\langle x, y\rangle \in EXT_I(S_I(\text{owl:someValuesFrom}))$ and $\langle x, p\rangle \in EXT_I(S_I(\text{owl:onProperty}))$, the supset relation is $CEXT_I(x) \supseteq \{u|\exists\langle u, v\rangle \in EXT_I(p)$ such that $v \in CEXT_I(y)\}$. Once we find out one existence, we can assert it as below.

```
(defrule OWL_someValuesFrom_supset
  (triple (predicate "owl:someValuesFrom") (subject ?x) (object ?y))
  (triple (predicate "owl:onProperty") (subject ?x) (object ?p))
  (triple (predicate ?p) (subject ?u) (object ?v))
  (triple (predicate "rdf:type") (subject ?v) (object ?y))
  =>
  (assert (triple (predicate "rdf:type") (subject ?u) (object ?x))))
```

In order to specify the subset relation, for $\langle u, v\rangle \in EXT_I(p)$, we first find out all possible types s of the individual v, and then check whether y is one possibility of s. If it fails, what we could do is to randomly assign one to belong to y, else $v \in CEXT_I(y)$ has been satisfiable.

```
(defrule OWL_someValuesFrom_subset
  (triple (predicate "owl:someValuesFrom") (subject ?x) (object ?y))
  (triple (predicate "owl:onProperty") (subject ?x) (object ?p))
  (triple (predicate "rdf:type") (subject ?u) (object ?x))
  (triple (predicate ?p) (subject ?u) (object ?v))
  (not (and (triple (predicate ?p) (subject ?u) (object ?o))
            (triple (predicate "rdf:type") (subject ?o) (object ?s))
            (test (eq 0 (str-compare ?s ?y)))))
  =>
  (printout t "Caution!" ?v " now is in " ?y crlf)
  (assert (triple (predicate "rdf:type") (subject ?v) (object ?y))))
```

owl:hasValue

Suppose $\langle x, y\rangle \in EXT_I(S_I(\text{owl:hasValue}))$ and $\langle x, p\rangle \in EXT_I(S_I(\text{owl:onProperty}))$, the set equation is $CEXT_I(x) = \{u | \exists \langle u, v\rangle \in EXT_I(p)\}$, and the translations are also easy.

```
(defrule OWL_hasValue_subset
  (triple (predicate "owl:hasValue") (subject ?x) (object ?y))
  (triple (predicate "owl:onProperty") (subject ?x) (object ?p))
  (triple (predicate "rdf:type") (subject ?u) (object ?x))
  =>
  (assert (triple (predicate ?p) (subject ?u) (object ?y))))
(defrule OWL_hasValue_supset
  (triple (predicate "owl:hasValue") (subject ?x) (object ?y))
  (triple (predicate "owl:onProperty") (subject ?x) (object ?p))
  (triple (predicate ?p) (subject ?u) (object ?y))
  =>
  (assert (triple (predicate "rdf:type") (subject ?u) (object ?x))))
```

owl:cardinality

Suppose $\langle x, y\rangle \in EXT_I(S_I(\text{owl:cardinality}))$ and $\langle x, p\rangle \in EXT_I(S_I(\text{owl:onProperty}))$, the subset relation is $CEXT_I(x) \subseteq \{u | card\{\langle u, v\rangle \in EXT_I(p)\} = y\}$. We can

compute the number of v using the Jess function "count-query-results". Error messages are thrown in case the result does not equal to y.

```
(defquery OWL_cardinality_query
    (declare (variables ?P ?S))
    (triple (predicate ?P) (subject ?S) (object ?O) ))
(defrule OWL_cardinality_subset
  (triple (predicate "owl:cardinality") (subject ?x) (object ?y))
  (triple (predicate "owl:onProperty") (subject ?x) (object ?p))
  (triple (predicate "rdf:type") (subject ?u) (object ?x))
  (test (<> ?y (count-query-results OWL_cardinality_query ?p ?u)))
  =>
 (printout t "Error!" ?x " has no " ?y " relating to " ?p crlf))
```

The supset relation is asserted by means of the "count-query-results" function.

```
(defrule OWL_cardinality_supset
  (triple (predicate "owl:cardinality") (subject ?x) (object ?y))
  (triple (predicate "owl:onProperty") (subject ?x) (object ?p))
  (triple (predicate ?p) (subject ?u) (object ?v))
  =>
  (if (= ?y (count-query-results OWL_cardinality_query ?p ?u)) then
   (assert (triple (predicate "rdf:type") (subject ?u) (object ?x)))))
```

3.4 The Pathology Rule Library

The rule library implements mainly the quality assurance guidelines provided by Institute of Pathology Charité. These guidelines are grouped in three levels of complexity:

Level 1 Overall quality criteria

Level 2 Lung and pleura pathology-specific criteria

Level 3 Disease-specific quality criteria

Our prototype rule base concerns mainly the first level of complexity and consists rules referencing the three main parts in a pathology report:

- macroscopy: the detailed description of the pathology compound should be available
- microscopy: optional rules concerning the microscopical properties of the compound
- diagnosis: a detailed description of the diagnosis decision process.

3.5 Lessons learned

We were concerned with 2 major problems in engineering the rule part of the pathology knowledge base: the first issue is directly related with the difficulties associated with scoping and limiting the domain the rules should describe. The second issue has a technological nature: as mentioned in the beginning of this section reasoning over heterogeneous Semantic Web knowledge bases containing OWL and rules (e.g. SWRL rules) is a hot topic in the Semantic Web community and there is no completely satisfactory solution to overcome the difficulties related to the trade-off between expressivity and decidability. The usage of decidable subsets of OWL and SWRL does not imply the decidability of the combined knowledge base. Further on, additionally restricting the two component representation paradigms does not solve the reasoning issue: should we use two reasoning engines to perform local inferences or should we translate the two languages to a common format? The absence of a native SWRL reasoning engine implies that at least the SWRL rule set should be transformed to a common logic programming language (Datalog, Jess etc.). In our implementation we used the second approach, due to the low expressivity of the OWL ontology library, which could be translated to a formalism such as Jess without any major loss of semantics (see Section 5.2 for details of the implementation).

Chapter 4

A RDF Format for Image Descriptions

4.1 General considerations

A efficient and well structured storing method is essential not only for the medical reports, also for the image descriptions.
For metadata description several frameworks were developed. One of the most common frameworks is the Resource Description Framework (RDF) introduced 1999 by the World Wide Web Consortium (W3C).
The RDF syntax uses among other the eXtensible Markup Language (XML). The interchangebility and propagation of XML and RDF fits our goal of an interchangeable and effective metadata format for medical image decription.

4.2 The RDF format for image description

For describing the pathological slides properly, three main metadata domains have to be covered. These are:

1. metadata describing the creation and content,
2. metadata describing the technical properties,
3. annotations for importent findings or objects.

To provide common readable image descriptions, the use of preexisting and approved vocabulary in RDF is mandatory. Several vocabularys for RDF exists. The following were found as best suitable for the three domains:

1. Dublin Core (DC),
2. Exchangeable image file format (EXIF) metadata,

3. Scalable Vector Graphics.

4.2.1 Dublin Core metadata

"The Dublin Core Metadata Initiative (DCMI) is an organization dedicated to promoting the widespread adoption of interoperable metadata standards and developing specialized metadata vocabularies for describing resources that enable more intelligent information discovery systems." [http://dublincore.org/about; 01-27-2005]
The Dublin Core ist a standard for metadata, describing digital objects. It consists of an set of 16 optional elements. We used the following 11 elements:

- title: a describing title for the image,
- subject: a brief description of the image content,
- description: a longer description of the image content,
- relation: the relation to the case number of the report,
- identifier: a unique identifier for the image resource,
- source: the URI of the image,
- publisher: the publishing institute,
- creator: the creating person,
- date: the creation date,
- format: the file format of the image,
- language: the description language.

4.2.2 EXIF technical information

Exchangeable image file format (EXIF) is a image file format used mainly by digital cameras. In addition to standard file formats like JPEG an metadata set is used. This metadata set is also available as RDF vocabulary. ¿From the wide range of elements of this metadata set we use the following for the description of technical image data:

- make: producer of the device for image recording,
- model: device model,
- xResolution: the number of pixels per ResolutionUnit in the ImageWidth direction,
- yResolution: the number of pixels per ResolutionUnit in the ImageHeight direction,

- resolutionUnit: the unit for measuring XResolution and YResolution,
- imageWidth: the number of pixels per row,
- imageLength: the number of rows of image data.

4.2.3 SVG for image annotation

For image annotation we needed a description language for shapes, points or areas within the image. The Scalable Vector Graphics (SVG) is a common open standard for two dimensional vector graphics. It was developed by the W3C and version 1.0 became a W3C recommendation on 2001-09-04. The SVG vocabulary is supplemented by DC elements for informations about the creation of the annotation (see Appendix 2).

Chapter 5

Implementation

5.1 Ontology Engineering

Details about the implementations can be found in the JavaDoc documentation of the tool, available at:

`http://www.inf.fu-berlin.de/inst/ag-nbi/research/swpatho/deutsch/javadocs.htm`

.

5.2 Rules Engineering: OWL2Jess

We implemented the transformation process of OWL files in Jess using a small (less than 100 lines of code) Java program and XSLT and draw inferences of OWL ontology and individuals with Jess. All expressive restrictions are handled with the help of error or caution messages, and the inferred assertions are helpful for the author to recognize possible extensions of the ontology. The RDF(S) and OWL semantics were coded manually as pre-defined Jess rules (see

`http://www.inf.fu-berlin.de/inst/ag-nbi/research/owltrans/`

).

.1 List of relevant UMLS Libraries

```
Alcohol and Other Drug Thesaurus
CPT
CRISP Thesaurus
Clinical Classifications Categories
Clinical Concepts
Clinical Problem Statements
DXplain
```

Digital Anatomist
Diseases Database
FDA National Drug Code Directory
Gene Ontology
Glossary of Clinical Epidemiologic Terms
Health Level Seven Vocabulary System
ICD-10
ICD-10 Am Engl
ICD-10 German
ICD-9-CM
ICPC 1993
ICPC 1993 German
ICPC2E 1998
ICPC2E 1998 Am Engl
ICPC2E 1998 Plus Am Engl
ICPC2E ICD-10 Relationships
LOINC
Library of Congress Subject Headings
MEDLINE Backfile
Master Drug Data Base
MeSH
MeSH German
MedDRA
MedDRA Am Engl
MedDRA Am Engl expanded
MedDRA expanded
Medical Entities Dictionary
Metathesaurus Names
Metathesaurus Source Terminologies
NCI SEER ICD Mappings
NCI Thesaurus
NLM Relationships
NLM RxNorm
Neuronames Brain Hierarchy
Online Mendelian Inheritance in Man 1993
Physician Data Query
Quick Medical Reference
Read Codes
Read Codes Am Engl
Read Codes Am Synth
Read Codes Synth
SNOMED 1982
SNOMED Intl 1998
Taxonomy from NCBI
UMLS Hierarchical Terms CPT
UMLS ICD-9-CM Terms

.2 Sample RDF Image Description

```
<?xml version="1.0" standalone="no"?> <rdf:RDF
xmlns:svg="http://www.w3.org/2000/svg#"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:exif="http://www.w3.org/2003/12/exif/ns#">
  <rdf:Description about="">
    <dc:title>Bronchialkarzinom</dc:title>
    <dc:subject>kleinzelliges Bronchialkarzinom</dc:subject>
    <dc:description>viel text bla bla bla</dc:description>
    <dc:relation>12599875</dc:relation>
    <dc:identifier>15778</dc:identifier>
    <dc:source>http://nbi.inf.fu-berlin.de/media/pathology/15778.jpg
    </dc:source>
    <dc:publisher>Charite Institut fuer Pathologie</dc:publisher>
    <dc:creator>Dr. P...</dc:creator>
    <dc:date>2004-04-11</dc:date>
    <dc:format>jpg</dc:format>
    <dc:language>de</dc:language>
    <exif:make>Company Xs</exif:make>
    <exif:model>Modell Y</exif:model>
    <exif:xResolution>600/1</exif:xResolution>
    <exif:yResolution>600/1</exif:yResolution>
    <exif:resolutionUnit>mm</exif:resolutionUnit>
    <exif:imageWidth>64000</exif:imageWidth>
    <exif:imageLength32000</exif:imageLength>
  </rdf:Description>
    <annotation>
      <dc:title>Mitose</dc:title>
      <dc:description>Tumorzelle in Mitose befindlich.
      </dc:description>
      <dc:creator>Dr. N...</dc:creator>
      <dc:date>2004-08-05</dc:date>
      <dc:language>de</dc:language>
      <svg:polygon fill="red" stroke="blue" stroke-width="10"
        points="350,75  379,161 469,161 397,215 423,301
            350,250 277,301 303,215 231,161 321,161" />
      <!--
        other forms e.g.:
        <svg:rect x="1" y="1" width="1198" height="398" fill="none"
        stroke="blue" stroke-width="2" />
        <svg:circle cx="600" cy="200" r="100" fill="red"
        stroke="blue" stroke-width="10"  />
        <svg:line x1="100" y1="300" x2="300" y2="100"
```

```
        stroke-width="5"  />
        -->
    </annotation>
</rdf:RDF>
```