Tackling the Ontology Acquisition Bottleneck: An Experiment in Ontology Re-Engineering

Sean Bechhofer [a] Aldo Gangemi [b] Nicola Guarino [b]
Frank van Harmelen [c] Ian Horrocks [a] Michel Klein [c]
Claudio Masolo [b], Daniel Oberle [d] Steffen Staab [d]
Heiner Stuckenschmidt [c,*] and Raphael Volz [d]

[a] *Manchester University, UK*
[b] *ISTC-CNR, Trento, Italy*
[c] *Vrije Universiteit Amsterdam, The Netherlands*
[d] *Institute AIFB, University of Karlsruhe, Germany*

Abstract

Ontologies form the backbone of the Semantics Web. They provide explicit descriptions of the conceptualization underlying available information. The effort involved in creating and maintaining ontologies is one of the most severe obstacles to a wide adoption of semantic web technology. In this paper, we address the problem of creating ontologies for the semantic and describe experiences with an approach that combines automated extraction of conceptual models from existing information sources with state-of-the-art methodologies and tools for ontological engineering. We further discuss tool interoperation based on semantic web technology as an enabler for successful ontology maintenance.

Key words: Ontological Engineering, Foundational Ontologies, Semantic Web, Tool Interoperation

1 Introduction and Motivation

Ontologies are set to play a central role in providing semantics for the web in terms of formalized, shared definitions of vocabularies that can be used to

* Corresponding author. Address: Vrije Universiteit Amsterdam, de Boelelaan 1081a, 1081HV Amsterdam, NL, email: heiner@cs.vu.nl, Tel.: +31 20 44 47 752, Fax: +31 20 44 47 653

access and exchange data between persons and applications in a meaningful way. The use of ontologies for improving information retrieval, sharing and integration has been investigated in many research projects and encouraging results have been achieved, providing evidence that large parts of the vision of the Semantic Web could actually be made reality. One of the most serious problems that still has to be solved, however, is the large scale creation and maintenance of ontologies that link to real data rather than being artificially designed models.

A number of methodologies have been proposed to deal with the problem of creating ontologies of complex domains [1–4]. All of these methodologies, however, require a high amount of manual engineering, making the effort of building ontologies time-consuming and error-prone. A number of techniques have been investigated recently that aim at automatically creating ontologies using different sources of information such as natural language, thesauri, instance data sets, or schema information from databases or XML documents. While some of these methods have already been used with some success, the resulting ontologies often do not adhere to design criteria for ontologies [5,6] and may even contain inconsistencies.

In this paper, we present some experiences gained in an attempt to overcome this problem, also known as the "acquisition bottleneck", using a combination of automatic ontology extraction techniques with state-of-the-art methods for ontology engineering and maintenance. Our experiences are based on a problem that can be assumed to be typical for the semantic web: the problem of enriching an existing database with ontological information describing the data and aligning it with a reference ontology that provides a common vocabulary for describing a wide range of information.

Our experiences result from a case study that was carried out as part of the European basic research project WonderWeb [1], which aims at the development of an ontology infrastructure for the Semantic Web. We applied different tools that have been developed in the project and provided interoperability between these tools using existing Semantic Web languages and standards.

Our work contributes to different areas of ongoing research on Web Semantics.

Ontological Engineering: We show that the combined use of automatic extraction methods with state-of-the-art ontology engineering approaches produces useful results. The combined approach can be expected to be less time consuming than manual approaches, and to produce better results than completely automatic methods.

Tool Support: We present tools that support the different steps of the engineering process starting from the extraction of an ontology from a database

[1] `http://wonderweb.semanticweb.org`

schema to the detection of changes between different versions of the extracted ontology. These tools provide valuable assistance for developers of ontologies in general, and of Semantic Web ontologies in particular.

Interoperability: We illustrate how different tools that have been developed completely independently of each other can interoperate using Semantic Web standards. We substantiate the claim that the use of Semantic Web languages such as RDF and OWL help to achieve interoperability and present a broker architecture for Semantic Web tools that has been developed in the project.

The paper is organized as follows. We start with an introduction of the use case scenario in section 2. We introduce the different steps taken in the use case as well as the data source and some basic technologies used. The main part of the paper consists of a more detailed discussion of the individual steps of the process. The main engineering steps are extraction (section 3), alignment (section 4) and refinement (section 5). We explain their role in the case study and present the methods and tools employed as well as the main insights gained. The remaining part of the paper is devoted to general issues connected with the process. These are the management of changes in the extracted ontology in the course of the process, discussed in section 6 and the interplay of the different tools via a broker architecture which is the topic of section 7. We conclude with a discussion of lessons learned and open issues in section 8.

2 The Use Case Scenario

The objective of the WonderWeb project is to develop a complete ontology engineering environment. This environment will include:

- Tools for creating and manipulating ontologies. E.g., tools for extracting ontologies from legacy resources, and ontology editing tools.
- Tools to help users manage and maintain ontologies. E.g., ontology comparison and version management tools.
- Basic services that can be used by tools and applications. E.g., persistent storage services and reasoning services.
- A library of foundational ontologies that can be used both as a starting point for the development of new ontologies, and as a means to align, validate, refine and enrich existing ontologies.
- A central server component that provides tool interoperation and uniform client access to the various tools and services.
- Methodologies that promote the development of high quality ontologies using the above tools, services and foundational ontologies.

In this paper we report on our experiences using the WonderWeb environment in a realistic ontology development scenario. In this scenario, we assume that the starting point will not be a "blank piece of paper", but will be an existing ontology/conceptual schema. Similar techniques would, however, be applicable to the development of a completely new ontology—in this case the suggested methodology would be to choose the most appropriate foundational ontology from the library, and use this as a starting point for the cleaning of the new ontology.

We anticipate that, in order to "boot-strap" the Semantic Web, it will be necessary to exploit legacy resources such as existing ontologies and conceptual schemas. In particular, we anticipate that automated techniques will be used both to derive basic ontologies and to annotate web resources using terms from ontologies (for example [7]). In our scenario, the initial ontology is automatically extracted from the relational schema of a database that might be used in a Human Resources (HR) department.

Extracted (or existing legacy) ontologies may be of relatively low quality, and provide relatively little detailed information about the classes and properties they describe. The quality of such ontologies, and hence their utility and reusability, can be greatly enhanced by aligning them with a carefully crafted and richly axiomatized foundational ontology that captures the high level structure of the relevant domain. Further refinement and enrichment of the ontology may also be beneficial at this stage, and if the ontology is to be used in applications, then ongoing development and maintenance is likely to be required over a period of time. [2]

In order to investigate the utility of the WonderWeb ontology engineering environment, we have applied it in the use case scenario illustrated in Figure 1. The scenario can be summarized as follows:

(1) Extract the HR ontology from a relational schema (using the OntoLift extraction tool).
(2) Save the ontology in a persistent store that is accessible to other tools and services (using the Sesame RDF storage component).
(3) Align the HR ontology with the DOLCE foundational ontology (using the OilEd ontology editor).
(4) Refine it (again using OilEd) in the light of DOLCE's foundational principles.
(5) Check the consistency of the ontology and compute implicit subsumption relationships (using the FaCT reasoning service).
(6) Track version changes and check their effect on applications (using the OntoDiff tool).

[2] It may also be advantageous to divide the ontology into modules, but we have not investigated this in our use case scenario.

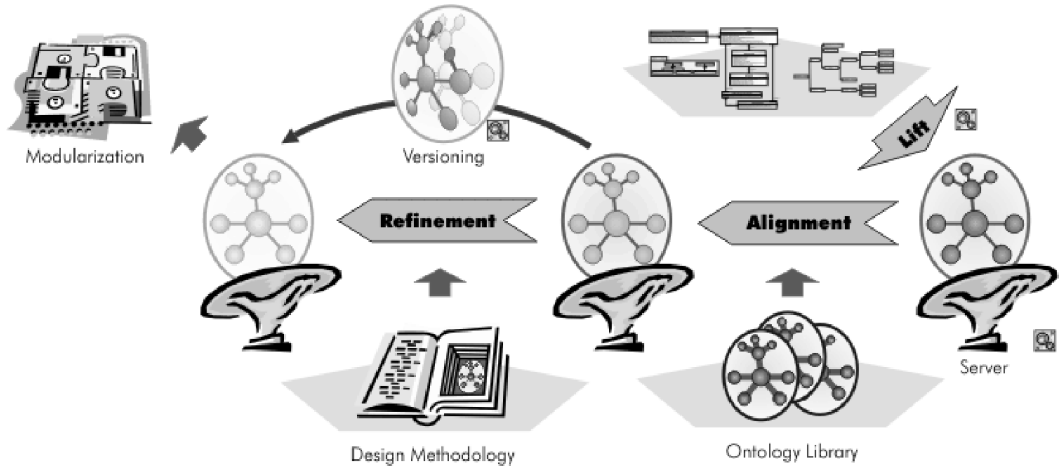

Fig. 1. Overview of the WonderWeb Case Study

In the first step , an initial "basic" ontology is captured by the system (see section 3). In our case we use extraction from a relational schema, but this could easily be substituted by extraction from other legacy resources (e.g., XML Schemas), importation of existing ontologies (e.g., RDF ontologies), or the development of an initial ontology design by users with limited ontology engineering experience (e.g., domain experts). In the next step, the ontology is linked to DOLCE by adding subsumption relationships between DOLCE classes and top level classes in the HR ontology (section 4. In the following step, the ontology is further refined, e.g., by improving class and property names, enriching class descriptions, and eliminating undesirable artifacts of the extraction process and a reasoner is used to help the ontology engineer check the effects of the alignment and enrichment steps (section 5). Finally, a version control tool is used to track the changes in the ontology and to check on the effects that changes are likely to have on applications of the ontology (compare section 6).

During ongoing development and maintenance of the ontology, steps 4, 5 and 6 may be repeated an arbitrary number of times. Different versions of the ontology can be stored for comparison purposes and to provide roll-back should it be necessary to revert to an earlier version of the ontology.

The KAON server (section 7 supports interoperation between the various components used in the scenario, and also provides uniform APIs for client applications. It is important to note that the KAON architecture is highly extensible, and that the components we use here are only an example of those that could be integrated using the server (in fact multiple editors, reasoners and storage components are already available).

3 Extracting an Ontology from Legacy Data

As mentioned in the previous section, we do not intend to build a new ontology from scratch but intend to reuse an existing conceptualization in form of a legacy database schema found in existing information systems. For our experiment we picked a small human resource database [3] (cf. Figure 2) as a starting point to build a corporate human resource ontology. Similarly we have chosen existing XML schemas [8] or UML diagrams [9] which underly the architecture of many modern information systems.

Our choice has three motivations. Firstly, the HR database schema is sufficiently simple to serve as an understandable example. Secondly, we assume that database schemas are present in great quantity in the modern information system landscape and therefore are the prominent type of schema encountered in practise. Thirdly, it's flat structure is representative for most database schemas.

Further, database schemas are often generated from an initial conceptual data model describing structures found in the domain under consideration. The actual logical database schema at hand is derived from such a conceptual model in a well defined process. As a consequence, logical database schemas still carry conceptual knowledge as we normally find it in an ontology. Extracting an ontology from a database schema can therefore be seen as a reverse engineering of the database into an explicit representation of the underlying conceptualization.

3.1 Schemas and Ontologies

Schemas often provide some description about the universe of discourse (UoD) in which a given information system operates. However, the role of schemas is different from ontologies, and therefore many constructors available for building schemas cannot be translated. For example, integrity and value constraints are typically used to ensure the validity of data for the given application. This underlines the purpose of schemas, namely to hold a description of data for a single application. Additionally, this description usually incorporates many choices that are conceptually irrelevant for the description of the UoD, e.g., choices made for reasons of application performance. Hence the conceptualization principle [10], which states that a conceptualization should consistently reflect the domain, is typically violated. Since conceptual representation is a non-purpose for schemas, schemas usually do not suffice for complex conceptual descriptions. For these reasons, one may not expect to extract very

[3] In fact it is the HR database schema that ships with the Oracle 9.2 installation.

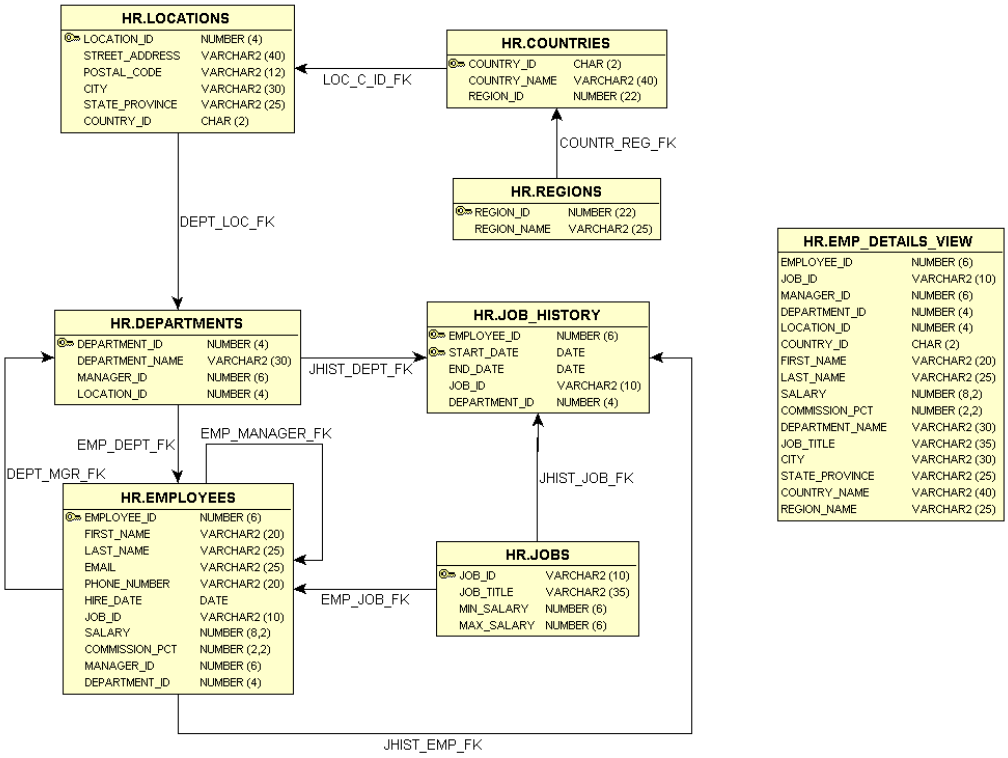

Fig. 2. The HR database schema (edge labels are identifiers of integrity constraints)

complex ontologies that meet high quality demands. We decided to output simple ontologies constituted by classes and properties (with domain and range constraints) that are expressible in both existing Semantic Web ontology language standards - RDF Schema [11] and OWL [12]. Interestingly, the lack of powerful conceptual construction mechanisms leads to so-called overspecification, i.e. the introduction of extra entities to capture certain facts, which has to be taken into account when lifting legacy schemas to ontologies.

3.2 Translating a relational schema

In order to extract an ontology from the HR database schema in Figure 2, we apply several translation heuristics. These heuristics operate on the underlying schema language of the relational database, viz. an extended relational model, which augments the usual formal definition of the relational model (e.g. in [13]) with additional constructs typically found in the data definition language (DDL) of databases, i.e. constructs which allow the statement of inclusion dependencies between different relations [14].

The translation heuristics themselves try to capture the semantics of a given database schema by translating relations and views to ontological classes and attributes to ontological properties. We can distinguish two types of attributes.

One corresponds to integrity constraints in the original schema, which express foreign key references to other relations. Such constraints are translated to OWL object properties with appropriate domain and range constraints. The remaining attributes are translated to OWL datatype properties. Several more sophisticated rules that go beyond these simple heuristics have been specified in [15] to deal with overspecifications, i.e. auxiliary relations in the original schema, and are intended to eventually remove redundant and irrelevant information. Similar translation heuristics can also be applied to capture the semantics of XML schemas and UML software specifications [9].

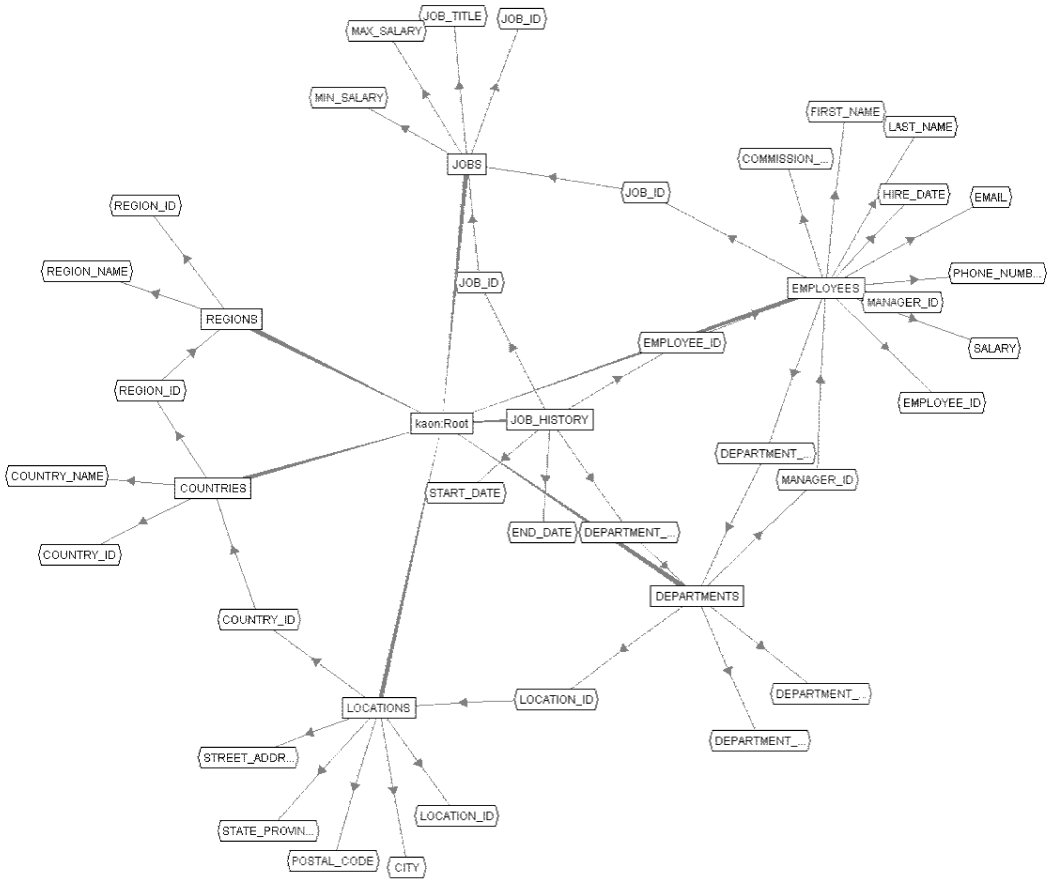

Fig. 3. The converted HR ontology

3.3 The HR Example

The application of the translation heuristics to the human resources (HR) schema allows us to extract 8 classes (one of which is constituted by a database view) and 51 properties (see figure 3).

Ten of these properties are object properties, which express class associations,

for example [4]:

```
ObjectProperty( http://www.test.de/HR/LOCATIONS/COUNTRY_ID
  domain( http://www.test.de/HR/LOCATIONS )
  range( http://www.test.de/HR/COUNTRIES ) )
```

These object properties are created after analyzing the inclusion dependencies stated for each relation via foreign keys. To label the properties we used the name of the source attribute; alternatively we could have used the name of the integrity constraint. We chose the former since the latter does not necessarily exist.

Unfortunately we cannot automatically identify whether the sole purpose of foreign key attributes in a relation is to express the association to other tables, or whether the attribute has a dual role as a carrier of conceptually relevant data. Hence, every attribute is additionally translated to a data property, for example:

```
DatatypeProperty( http://www.test.de/HR/COUNTRIES/COUNTRY_ID
  domain( http://www.test.de/HR/COUNTRIES )
  range( rdfs:Literal) )
```

Figure 3 depicts the resulting human resources (HR) ontology as a snowflake diagram. In this diagram top level classes (squares) are found in the center and subclasses are on the perimeter. Properties (bevelled squares) usually constitute the leafs of the graph and are connected with their domain classes. Object properties additionally may link to their range class.

The distribution of classes and properties has a ratio of one class per six properties. This is symptomatic of the nature of the source metadata: relational tables contain a lot of redundant relationships resulting from the transformation of the database schema into tables. Other features of HR are clearly related to its nature, for example the high number of concrete data types, such as strings, numbers, etc. The existence of so many literals in HR is due to the amount of foreign keys that require identifiers in the database. This is also a typical difference between a database schema and an ontology: in a database schema people tend to use concrete data types to represent domain items (typically, attribute values, in a ER scheme) whose ontological nature and structure is not relevant from the point of view of querying the database. In a foundational ontology, on the other hand, all domain items must be properly described in terms or basic ontological primitives. Attribute values become therefore first class citizens, and cannot be considered just as strings or numbers.

[4] We use the normative ascii notation for OWL DL defined in [16]

4 Grounding into a Foundational Ontology

The ontology extracted from the schema of the HR database is not yet suited to provide the basis for semantic information exchange. In particular it suffers from weak ontological commitments that are often not explicated but just based on the choice of concept names and the existence of links between these concepts. In order to make the ontology comparable to other models and clarify its underlying assumptions, we have to put some effort into a manual improvement of the model. For this purpose, we largely rely on ONIONS, the methodology described in [17]. The improvements involved:

- linking HR to an existing foundational ontology;
- refining and clarifying the basic ontological assumptions;
- revising the naming policies.

We will not describe the methodology in detail here, instead we prefer to explain the work carried out in this use case. In brief, the first two tasks have been:

(1) **choosing a foundational ontology**, which constitutes a principled, domain independent ontology. In the use case we employed DOLCE, which is a module of the WonderWeb ontology library. This allows us to understand and position the implicit intended meaning of HR.
(2) **linking to the foundational ontology**, by creating subsumption links from HR classes and properties to DOLCE according to the meaning of DOLCE classes and properties, as specified by means of DOLCE's axioms.

Foundational ontologies are characterized by small but richly axiomatised class hierarchies. They explicate ontological assumptions made by the foundational ontology and constrain the structure of domain ontologies. Basing a domain ontology on a suitable foundational ontology significantly improves its quality, and promotes good design.

Even with the aid of sophisticated editors and reasoners, the development of ontologies is a difficult task. Moreover, ontology development is often carried out by domain experts' who may not be skilled ontologists or logicians. The result is often low quality ontologies that poorly capture the experts' intuition about the domain and which are unlikely to facilitate integration or reuse. Hence it is promising to start the restructuring of the HR ontology by building on the firm basis of a foundational ontology.

The WonderWeb toolkit includes a set of foundational ontologies covering a wide range of application domains. Each of these ontologies provides a carefully crafted taxonomic backbone with a sound high level structure that can

be used as the basis for the development of more detailed domain ontologies.

In our scenario we chose the DOLCE [18] ontology as a basis for the alignment with HR. DOLCE is basically maintained in a first order logic with modal extensions (S5), but versions of it exist in KIF and other languages as well.

For the use case described here, we used DOLCE light which is an OWL version of DOLCE, since fully axiomatized DOLCE is not expressible in OWL. DOLCE-Lite contains some of the axioms present in the full DOLCE, but tends to have a wider coverage, by reusing parts of other ontologies (e.g. WordNet) that have been merged with the basic DOLCE. The version used for the experiment contains 80 classes, 80 properties (incl. inverses), 84 restrictions, and 24 disjointness axioms. See for example the class Political Geographic Object from DOLCE-Lite in OWL:

```
Class (dolce:Political-Geographic-Object partial
 dolce:Non-Physical-Place
 restriction(
   dolce:PHYSICALLY-DEPENDS-ON
   someValuesFrom dolce:Geographical-Object ) )
```

Expressive ontologies such as DOLCE can be used for meaning negotiation and explanation:

- to *negotiate meaning* between (human or artificial) agents belonging to different (possibly related) communities;
- to *establish consensus* in a community that needs to adopt a new term;
- to *explain the meaning* of a term to somebody new to the community.

These tasks require the explicit representation of *ontological commitment* in order to exclude terminological and conceptual ambiguities bound to unintended interpretations. Therefore, a rich axiomatization (in addition to an adequate informal documentation) is required. This is the very reason for maintaining DOLCE in full first order logic.

The alignment procedure aimed at finding subsuming classes or properties in DOLCE-Lite for HR elements. For example, hr:JOBS, whose intended meaning can be reconstructed by looking at the attached properties (e.g., hr:MAX-SALARY), is subsumed by dolce:Activity, while hr:EMPLOYEES in turn is subsumed by dolce:Social-Role. Less obvious is determining that property hr:JOBS_ID, holding between hr:EMPLOYEES and hr:JOBS, is subsumed by dolce:PARTICIPANT-IN; this implies recognizing that employees are *endurants* (loosely speaking, objects), jobs are in this context *perdurants* (loosely speaking, processes or events), and a generic formal relation of *participation* holds between these two basic categories. This linking procedure is clearly non-trivial, but brings significant benefits in terms of the quality, maintainability and reusability of the

ontology (as we will see in the following section).

In the alignment procedure, we have found that much meaning in HR is implicit in the names used for concepts and relations. Moreover, most names are used with multiple meanings, some classes have both names and identifiers, and a special class derived from a view on the database contains redundant information. These issues are addressed in the refinement procedure.

Class restrictions are particularly needed for developing and applying foundational ontologies. Consider for example the property employee_id from HR in OWL [5]:

```
ObjectProperty( http://www.test.de/HR/EMPLOYEES/EMPLOYEE_ID
  domain( http://www.test.de/HR/EMPLOYEES )
  range( rdfs:Literal ) )
```

In the original ontology, the range of the relation was defined in a very generic way, that hardly carries any meaning: indeed, a literal is just a sequence of characters without further structure or interpretation. If we want to make this concept comparable with other pieces of information on a semantic level, we have to link it to an upper level ontology. This requires complex interactions with the model that go beyond just drawing links between elements. This process actually corresponds to a refinement of the extracted ontology not only in terms of its content, but also in terms of the vocabulary used. We discuss such refinement process in the following section.

Before proceeding, let us remark that, instead of DOLCE, we might have chosen a different general ontology, reflecting different commitments and purposes. DOLCE however was particularly good for this experiment, since its careful axiomatization forces the user to clarify his/her ontological choices before making a link. However, the aim of WonderWeb is to have other modules - besides DOLCE - in a library of foundational ontologies, to account in a systematic way for different commitments and purposes. Rather than having just a flat repository of logical theories, the key idea is to link them together depending on formal relationships among the fundamental ontological options they adopt. Rationales and alternatives underlying the different ontological choices should be made as explicit as possible to make the consequences of choosing a certain foundational ontology explicit. In this way, a network of different but systematically related ontologies becomes available, and people working on applications could easily find those modules that better capture

[5] For the purposes of readability, examples here are given using the proposed OWL abstract semantics. See `http://www.w3.org/2001/sw/WebOnt/` for details. Through the examples, the namespace `dolce` refers to `http://www.loa-cnr.it/ontologies/DOLCE-Lite`, while `rdfs` refers to `http://www.w3.org/2000/01/rdf-schema`

their ontological needs. Such an architecture, we believe, will contribute to make people (and computers) commit to common ontological agreements as well as *understand the reasons of disagreement*, which seem to us to be more effective than enforcing interoperability by means of a single overarching ontology.

5 Refining the Ontology

As we have already seen at the end of the last section, there is a need to carry out modifications of the extracted ontology that go beyond the creation of links to other ontologies. These modifications not only concern more complex links to upper level categories, but also adjustments inside the extracted ontology. These adjustments are often concerned with fixing problems related to the different nature of ontologies and relational database schemata that were not captured by the heuristics of the extraction method.

We assume that the necessary corrections are made by a human knowledge engineer experienced enough to see and understand the remaining problems. Even so the complexity of the refinement task makes it essential to provide the knowledge engineer with tools that support the execution and validation of changes in the ontology. In particular, we have to provide an editor that allows the engineer to inspect the ontology and insert refinements. As mentioned before, this includes the possibility of adding richer constraints to classes than provided by RDF schema. Further, as soon as complex constraints are added to the ontology there is the risk of inconsistency. Providing reasoning support for checking the consistency of a model is a way to deal with this problem.

In the following, we describe the tools used to support the refinement process in the case study, as well as some of the refinements actually made.

5.1 Using OilEd to Refine the HR Ontology

In the refinement procedure, elements of the HR ontology are transformed in order to explicate as much ontological information as possible in a consistent form. This involves three steps

(1) **reorganizing** the taxonomy of HR.
(2) **enriching** HR with restrictions derived from those existing in DOLCE.
(3) **renaming** HR elements with cognitively transparent names. Moreover, class and property names and labels are changed following conventional naming policies.

For example, hr:STREET-ADDRESS is a property that relates hr:LOCATIONS to rdfs:Literal in the extracted HR ontology. But rdfs:Literal is ontologically opaque, since it simply refers to an information object, not to its meaning. There are situations in which a property must hold for literals (e.g., the (dataType)property dolce:NAME), but in our example we may want to commit to addresses in our domain of discourse. In this case we have refined hr-ref:STREET-ADDRESS to be a property of (the newly introduced hr-ref:ADDRESS, which is a subclass of dolce:NON-PHYSICAL-PLACE.

There are many reasons for having such a commitment. On the one hand, we may want to share the concept of an address with other databases for the sake of interoperability: this cannot be done consistently based on the similarity of the names in relational tables. On the other hand, we may want to reason about the internal structure of addresses, and having them as strings prevents this investigation, since the internal structure of a string (a literal is composed of characters) is completely different from the internal structure of the intended meaning of an address (street, number and zip codes denote places).

Some properties from HR have been deleted during refinement because they have a structural rather than ontological function. Their place has been taken by additional restrictions on classes. For example, hr:JOB_ID holding between hr:EMPLOYEES and hr:JOBS has been transformed into a restriction for the class hr-ref:EMPLOYEE claiming that there exists at least one of the values of the property dolce:PARTICIPANT-IN in the type hr-ref:JOB:

```
restriction(
  dolce:PARTICIPANT-IN
  someValuesFrom http://www.test.de/HR-REM/JOB)
```

hr:EMPLOYEES has also been renamed to hr-ref:EMPLOYEE (singular form is conventionally preferred). Another, more relevant example of renaming is the property hr:EMPLOYEE_ID (holding between employees and job histories), changed to hr-ref:EMPLOYEE_JOB_HISTORY. Polymorphic properties have been treated by either renaming or removing one of their projections. Removing is the natural choice when polymorphism couples with ontological redundance. For example, hr:JOB_ID has two projections: hr:EMPLOYEES to hr:JOBS, and hr:JOB_HISTORY to hr:JOBS. The first projection has been transformed into a restriction on hr-ref:EMPLOYEE (see above), while the second has been transformed into a restriction for the class: hr-ref:JOB that reuses the property dolce:TEMPORAL-LOCATION:

```
restriction(
  dolce:TEMPORAL-LOCATION
  someValuesFrom http://www.test.de/HR/JOB_HISTORY)
```

Some classes are related to both names and identifiers (literals that are used as foreign keys). Properties having identifiers in their range have been deleted. Properties having names in the range have been considered individually as shown. Some of them have been preserved when the concrete data type expressed by the literal belongs to a so-called quality region. For example, the property: hr-ref:JOB_MAX_SALARY ranges on numbers that need to be computed by algorithms external to the reasoner [19]. From the viewpoint of DOLCE, a salary for a job should be represented as a dolce:Abstract-Region with associated metrics that includes monetary units and number series. This is useful e.g. for computing a salary in different monetary units. The resulting range restriction for the hr-ref:JOB_MAX_SALARY could be refined as follows:

```
range
 rdfs:Literal
 restriction(
   dolce+:AMOUNT-OF
   someValuesFrom
     dolce+:Monetary-Unit
     restriction(
       dolce+:UNIT-OF
       someValuesFrom hr-ref:Salary))
```

The final refined and renamed version of HR consists of 14 classes, 18 properties, and 23 restrictions, thus getting a distribution of ontological elements more akin to that of foundational ontologies.

A final remark should be made about the relevance of DOLCE for preventing or highlighting common modelling mistakes. For example, the class:hr-ref:MANAGER can be assigned with plausible reasons to two different superclasses: hr-ref:EMPLOYEE and hr-ref:JOB. The first one is subClassOf dolce:Social-Role, while the second is subClassOf dolce:Activity. But in DOLCE, dolce:Social-Role is disjoint from dolce:Activity, since a role is a so-called endurant (an object or substance-like entity), while an activity is a so-called perdurant (either an event, a process, or a state). Endurants can participate in perdurants, but they are mutually disjoint.

In this case the reasoner would correctly detect an inconsistency and the modeler will have to choose one interpretation or the other. In the HR case, hr-ref:MANAGER should be subsumed by hr-ref:EMPLOYEE only, given the set of properties it is used in (no other clues are actually available from the database structure or documentation). In particular, manager refers to the subset of employees who *participate in* the job of managing.

Another frequent source of inconsistency in realistic modelling practice comes from misplaced restrictions. For example, a similar confusion about the mean-

ing of the concept of a job would make it plausible to add a restriction to hr-ref:JOB stating that jobs participate in activities:

```
restriction(
  dolce:PARTICIPANT-IN
  someValuesFrom dolce:Activity)
```

but the interpretation of hr-ref:JOB requires it being a subclass of dolce:Activity, and activities cannot participate in other activities (they can be part of other activities, or precede them, etc.). This useful constraint comes from a restriction inherited by hr-ref:JOB from dolce:Activity stating that only endurants can participate in perdurants (like jobs):

```
restriction(
  dolce:PARTICIPANT
  allValuesFrom dolce:Endurant)
```

We envisage extending the experiment by using the refined HR ontology to allow interoperation between two heterogeneous databases on human resources.

These examples show that refinement according to a richly axiomatized foundational ontology can not only improve the structure of an ontology, but can also allow errors to be detected as a result of the elimination of unwanted models. In this case, the axiomatization of dolce:Social-Role and dolce:Activity allow errors to be detected, e.g., when a class is asserted to be subsumed by both, or when inappropriate restrictions are added.

We would like to stress that the axioms that allowed to check the consistency and the soundness of the HR ontology are simply inherited from DOLCE, thus they are available to ontology building projects for any domain.

5.2 Editing and Reasoning with OilEd

For the purposes of the scenario, there are a number of tasks that an editor needs to support. These include:

(1) importing, inspecting and editing the extracted ontology (as an RDF Schema);
(2) exporting the ontology in OWL;
(3) importing the foundational ontologies (DOLCE-Lite);
(4) making assertions involving the relationship between concepts in the extracted ontology and DOLCE-Lite;
(5) providing explicit definitions of concepts in terms of other concepts from the ontology (and foundational ontologies);

(6) invoking a reasoner in order to determine whether the resulting refined ontology is consistent.

Points 1 to 4 are important for the alignment process, as the domain ontology concepts must be related to the foundational ontology concepts. Points 5 and 6 are crucial during refinement, as explicit descriptions of the domain concepts are introduced.

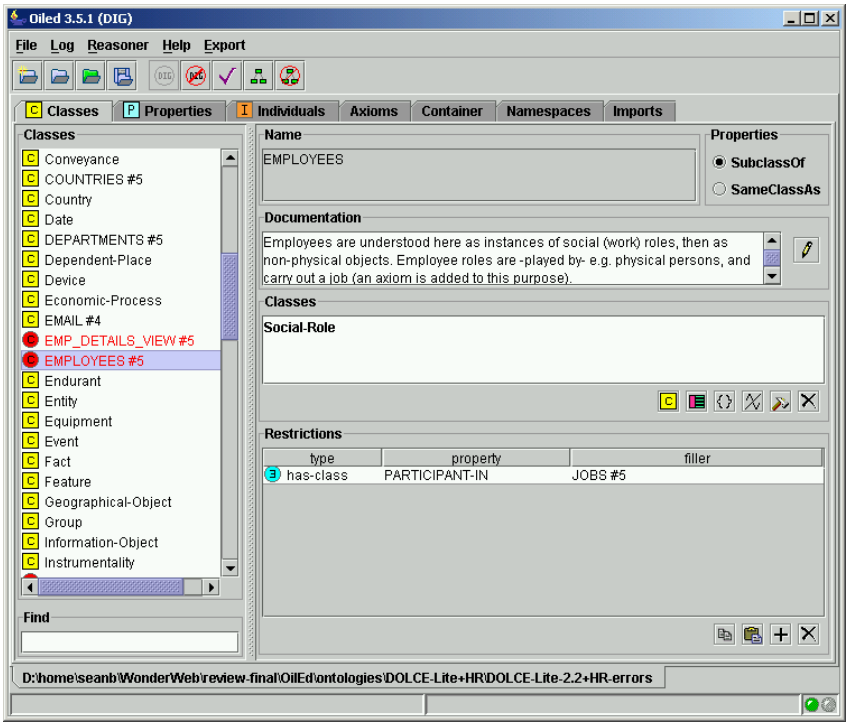

Fig. 4. OilEd Screen Shot

The experiment described here was carried out using the OilEd ontology editor [20]. OilEd allows the user to create, edit and manipulate OWL ontologies. OilEd is not a fully fledged editing environment, but instead provides a rather lightweight tool for ontology editing. The functionality is sufficient, however, for this demonstration scenario. OilEd has been discussed elsewhere [20], so we will not go into detail here other than to describe the key aspects of the tool and its place within the ontology construction scenario described in this paper.

An important aspect of OilEd is that unlike current versions of existing frame-based ontology editors such as Protégé or OntoEdit, it supports the full expressive power of OWL DL. Thus users can produce concept descriptions that make use of, for example, arbitrary boolean combinations of expressions and explicit quantification. The ability to use this expressiveness is exercised during the alignment and refinement processes of the scenario. In particular, the refinement stage requires that the modeler can explicitly articulate the relationships between the domain concepts in the HR (domain) model and those

concepts from the DOLCE-Lite (foundational) upper ontology. Indeed, simply being able to represent the DOLCE-Lite presentation in OWL itself requires that the tool supports a large subset of OWL's expressivity.

As richer and more complex relationships are introduced between the domain concepts, support for the user becomes important. During the process of ontology refinement as described in below, we can make use of a *reasoner* such as FaCT [21] to assist the ontologist. Tasks that can benefit from reasoning include:

- checking the consistency of concept definitions;
- producing an inferred concept hierarchy based on the definitions given.

As we see in the scenario, the first of these can prove particularly useful when the ontology has been richly axiomatised. Figure 4 shows a screenshot of the editor. The editor is telling us that the class hr:EMPLOYEES class has been found to be inconsistent (due to the restrictions placed on it).

OilEd interacts with a Description Logic reasoner hosted by a server (via a protocol known as DIG [6] [22]), allowing a modeler to perform the tasks described above.

6 Managing Ontology Change

Ontologies will inevitably change over time. Each step in the described engineering process involves changes in the ontology. Moreover, ontologies that are build on real data or applications could also change because of changes in the real world. Serious use of ontologies requires advanced methods for change management, as the changes could cause problems when other ontologies refer to definitions in the evolving ontology.

In the next paragraphs, we will describe a methodology that helps to localize the changes and predict the effect for different tasks. We illustrate this with examples from the changes that occur during the refinement steps, although the methodology can also be applied to changes in other phases of its life-cycle. The kernel of the methodology is a three-steps procedure: 1) finding the differences between two distinct versions of the ontology, 2) listing the specific changes for individual concepts and relations, and 3) determining the impact of these changes for a specific use of an ontology. The next sections describe these steps.

[6] See `http://dl.kr.org/dig` and `http://dl-web.man.ac.uk/dig` for further details about the activities of the Description Logic Implementation Group and the DIG protocols.

6.1 Finding Differences

To find changes in ontologies, we have developed a mechanism and a tool to compare ontologies. This change detection mechanism is described in [23]. The algorithm that we developed works for all ontology languages that can be represented in the RDF data model [24], including RDF Schema and OWL. For each changed definition, it produces a list of change operations that are necessary to transform the old version into the new version.

Changes in DOLCE+HR: Each of the steps in the alignment and refinement phase involve some typical changes. We will briefly summarize them and show some changes that are typical for a specific step.

In the alignment phase, the concepts and properties in the extracted HR ontology are connected to concepts and properties in the DOLCE ontology via subsumption relations. For example, the concept 'Departments' from the HR ontology is made a subclass of 'Social-Unit' in DOLCE.

During the reorganization and enriching step (1 and 2) in the refinement phase, many changes are made. Some property restrictions are added, and some additional concepts and properties are created to define the HR concepts more precisely. For example, the concept 'Administrative-Unit' is introduced as a new subclass of 'Social-Unit', and the concept 'Departments' is made a subclass of it. Also, the range of the property 'email' is restricted from 'Abstract-Region' to its new subclass 'Email'.

During the renaming step (3), a number of concepts and properties are renamed to names that better reflect their meaning. For example, 'Departments' is renamed to 'Department' (singular), and the two different variants of the relation 'manager_id' are renamed to 'employee_manager' and 'department_manager'.

Finally, all properties and concepts that are not necessary anymore are removed and transformed into property restrictions. For example, the property 'employee_email' is deleted and replaced by an existential restriction in the class 'Employee' on the property 'abstract_location' to the class 'Email'.

6.2 Specifying Change Operations

For each changed definition, the detection mechanism produces a list of change operations that are necessary to transform the old version into the new version. To standardize the description of changes, we have developed an ontology of all possible change operations for an OWL-lite ontology. An actual description

of a change between two versions of an ontology can be seen as an instantiation of the ontology of change operations.

The change ontology is extendable to other knowledge models. We have chosen the OWL-lite model because of its simplicity and the central role of OWL in the WonderWeb project. A snapshot of the change ontology can be found online. [7] Apart from *atomic change operations* — like add range restriction or delete subclass relation — our change ontology also contains some *complex change operations*, which consist of multiple atomic operations and/or incorporate some additional knowledge. The complex changes are often more useful to specify effects than the basic changes. For example, for operations like concept moved down, or range restricted, we can specify the effect more accurately than for the atomic operations subclass relation changed and domain modified [8] .

The tool that we developed prints the change operations in the upper left corner of each marked change. The figure shows that the old definition of 'Departments' can be transformed into the new definition with three change operations: 1) change the superclass relation from 'Social Unit' to 'Administrative-Unit', 2) change to comment, and 3) add a specific property restriction. Note that the first one is actually a complex operation concept moved down, because 'Administrative-Unit' is a subclass of 'Social Unit'. In the future, the tool should be able to export these changes as RDF instance data for the change ontology.

6.3 Determining the Effect

Now we have detected the change operations that are required to transform the old version of the ontology into the new version, we look at the effect of the change operations for a *specific usage* of the ontology. This is important, as some changes might affect one task, while other tasks are unaffected. For example, if an ontology is only used as a vocabulary to retrieve instance data, the addition of a class does not have any effect. However, if the set of subclasses of a specific class is queried, the answer might change.

As an example, we will now show how we heuristically determine the effect on classification reasoning for individual concepts and relations. We have done something similar for the effect on data interpretation.

Assuming that C represents the concept under consideration before and C' the concept after the change there are four ways in which the old version C may relate to the new version C':

[7] `http://ontoview.org/changes/1/3/`
[8] Compare `http://wonderweb.man.ac.uk/deliverables/D20.shtml`.

(1) the meaning of concept is not changed: $C \equiv C'$ (e.g. because the change was in another part of the ontology, or because it was only syntactical);
(2) the meaning of a concept is changed in such a way that concept becomes more general: $C \sqsubseteq C'$
(3) the meaning of a concept is changed in such a way that concept becomes more specific: $C' \sqsubseteq C$
(4) the meaning of a concept is changed in such a way that there is no subsumption relationship between C and C'.

We describe what theoretically could happen to a concept as result of a modification in the ontology. To to so, we have determined the effect for all possible change operations that we distinguish in the 'specifying changes' phase. Table 1 contains some examples of operations and their effect on the classification of concepts. The table only shows a few examples, although our full ontology of change operations contains around 120 operations.

	Operation	**Effect on C**
1	Attach a property-retriction to class C	Specialized
2	*Complex:* Change the superclass of class C to a class lower in the hierarchy	Specialized
3	*Complex:* Restrict the range of a property S (*effect specified for all classes C that have a property restriction with S*)	Specialized
4	Remove a superclass relation of a class C	Generalized
5	Change the class definition of C from primitive to defined	Generalized
6	Add a class definition A	Unknown
7	*Complex:* Add a (not further specified) subclass A of C	No effect

Table 1
Some ontology change operations and their effect on the classification of concepts in the hierarchy.

If we apply this to our example, we can only give a useful characterization of the effect on some of the concepts. For example, the concept 'Departments', underwent several changes during the whole process: its superclass has changed to a subclass of the original superclass (change 2 in Table 1) but there are also some property restriction removed. Both changes have an opposite effect. As a result, we have to characterize the effect of the change as "Unknown". On the contrary, the effect on the relation 'department_manager', is clear: the relation is renamed from 'manager_id' — which has no conceptual effect — and the range is changed form 'Employee' to 'Manager'. Because 'Manager' is a subclass of 'Employee', this change makes it more specific (the argument is similar to change 3 in Table 1 applied to properties instead of concepts).

7 Tool Interoperation

As we have seen in the previous sections, several software modules have to be technically integrated in order to realize the human resource scenario. We address this problem by using the KAON SERVER — WonderWeb's main organizational unit and infrastructure kernel. In subsection 7.1, we generally describe its idea and architecture, whereas subsection 7.2 revisits our use case and shows how the KAON SERVER can be used to facilitate technical interoperation in this particular case.

7.1 KAON SERVER

Building a complex Semantic Web application typically requires more than a single software module. Ideally the developer of such a system wants to easily combine different — preferably existing — software modules. So far, however, such integration had to be done in an ad-hoc manner, generating a one-off endeavour, with few possibilities for reuse and future extensibility of individual modules or the overall system.

Hence, there is a need for an infrastructure that facilitates reuse of existing modules, e.g. ontology stores, editors, and inference engines and, thus, the development and maintenance of comprehensive Semantic Web applications, an infrastructure which we call *Application Server for the Semantic Web (ASSW)* [25,26]. For WonderWeb, we developed *KAON SERVER*, a particular Application Server for the Semantic Web, which is part of the Karlsruhe Ontology and Semantic Web Tool Suite (cf. http://kaon.semanticweb.org)

In the terminology of [27], KAON SERVER would be called a *Broker*, i.e. transactions from the client application to the software module are always handled and possibly modified by interceptors. Its architecture basically uses a Microkernel and component approach. The Microkernel offers a minimal functionality of managing, i.e. starting, stopping and initializing components. Existing software modules have to be made deployable [9] in order to be managed by the Microkernel. Thus, in our terminology, a software module becomes a *Component*. In order to distinguish between components that are of direct interest to the developer and components providing functionality for the Application Server itself (e.g. connectors or the registry), we call the first *Functional Components* and the latter *System Components*.

[9] We use the word deployment as the process of registering, possibly initializing and starting a component to the Microkernel.

Apart from the cost of making existing software deployable, this approach delivers several benefits. By making existing software modules deployable, one is able to manage them in a centralized infrastructure. As a result, we are able to deploy and undeploy components in an ad hoc manner, reconfigure, monitor and possibly distribute them dynamically. Proxy components can be developed for *External Modules*, viz. modules that cannot be made deployable. A client can semantically discover the component it is in need of and interceptors (see below) are a powerful mechanism to provide security, transactions, semantic interoperation to name but a few. The following paragraphs detail the architecture which is depicted in Figure 5.

Connectors *Connectors* are system components that send and receive requests and responses over the network, e.g. via Java Remote Method Invocation (RMI). Counterparts to a connector on the client side are *Surrogates* for components that relieve the application developer of the communication details, similar to stubs in CORBA.

Management Core The Management Core comprises the Microkernel as well as several system components. First, the *Registry*, which is a simple ontology store, manages ontological descriptions of the components and facilitates their discovery for the application developer. Second, another system component called *Association Management*, allows us to put associations into action. The management ontology defines associations between components that might have to be operationalized. E.g., a component can rely on another or there might be an association "receivingEventsFrom" between two components. Third, the *Component Loader* facilitates the deployment process for the developer. Components and their libraries can be stored in an archive along with a corresponding ontological description. The Component Loader takes the URL of that archive as an argument, reads the description, enters it in the registry and puts possible associations into action by applying the Association Management component.

Interceptors *Interceptors* are software entities that monitor a request and modify it before the request is sent to the component. For example, a semantic interoperation interceptor might seamlessly translate between different Semantic Web languages used by a client and a deployed ontology store.

Functional Components RDF stores, ontology stores etc., are finally deployed to the Microkernel as Functional Components. In Figure 5 we also distinguish Functional and System Components visually, even though there is no technical difference.

7.2 The use case scenario revisited

Throughout our use case scenario, several software modules have been used to solve certain tasks (cf. section 2). This subsection shows how their interop-

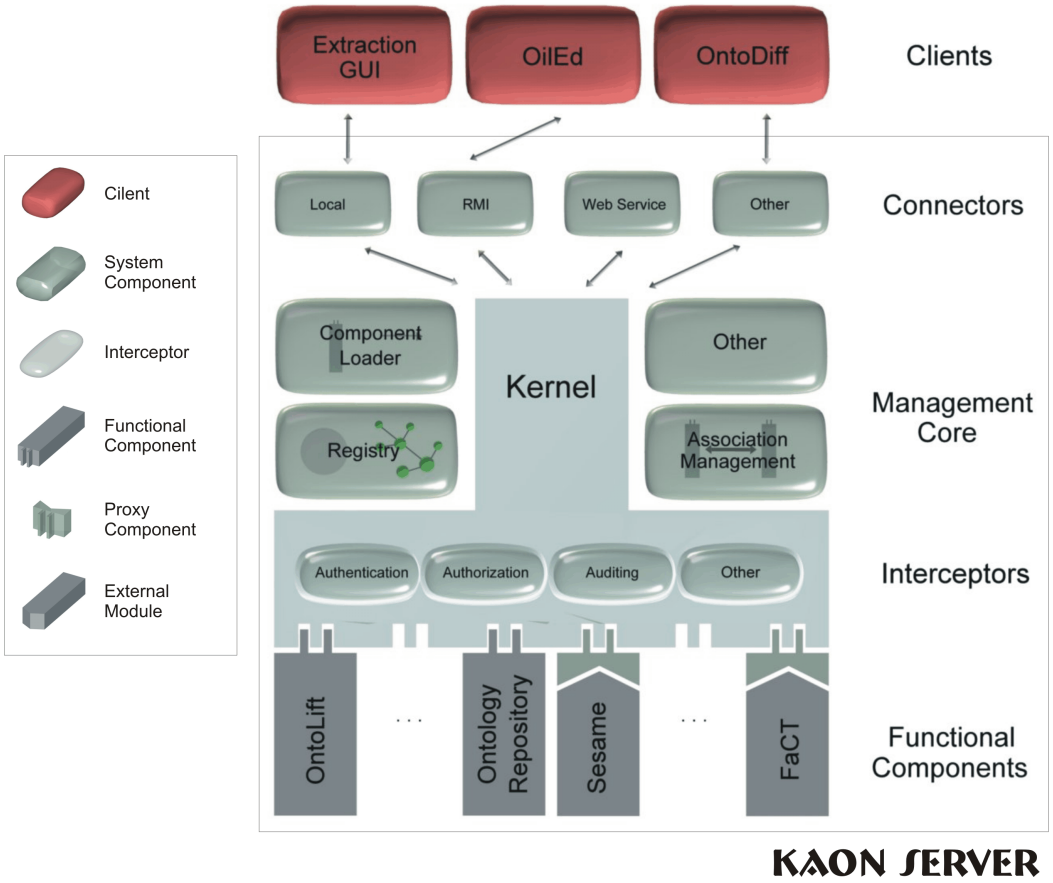

Fig. 5. KAON SERVER's architecture.

eration can be facilitated by making use of the KAON SERVER. First of all, Table 2 provides an overview about the modules and their role in the KAON SERVER.

Step and Functionality	Software Module	Role in the KAON SERVER
1. Ontology Extraction	OntoLift	Functional Component
2. Ontology Storage	Sesame	External Module
3. Ontology Alignment	OilED	Client
4. Ontology Refinement	OilEd	Client
5. Consistency Check	FaCT	External Module
6. Tracking of change	OntoDiff	Client

Table 2
The software modules' roles in the KAON SERVER.

OntoLift has been developed especially for the project as a functional component. Additionally, there is a GUI visualizing the extraction for the user which in turn becomes a client of KAON SERVER. Sesame remains an external module and we have developed a corresponding proxy component. Sesame

could also be used in order to store the set of foundational ontologies in step 3. However, any other ontology repository, i.e., a simplistic ontology store, could be made deployable for this step. In step 4, OilEd acts as a client to KAON SERVER by making use of the the Sesame storage and the FaCT inference engine. The latter remains an external module as it is written in another programming language and is made deployable by a proxy component. Finally, OntoDiff becomes a client and allows the user to track changes of ontologies stored in the deployed Sesame. Figure 5 depicts the KAON SERVER's architecture with the clients and components mentioned above.

In the remaining paragraphs we describe how the interaction between clients and KAON SERVER takes place. We assume that an instance of the KAON SERVER is up and running with all the necessary components. For the sake of brevity, we only describe OilEd's interaction — the Extraction GUI's and OntoDiff's are similar. Figure 6 shows what is happening on the client-side in a coarse sequence diagram. A client typically uses surrogate objects (usually called Remote*Component*) that reveal the same API as their corresponding components and just handle the communication details. The surrogates relay requests over the network where they are received by connectors and routed through the Microkernel to the appropriate component.

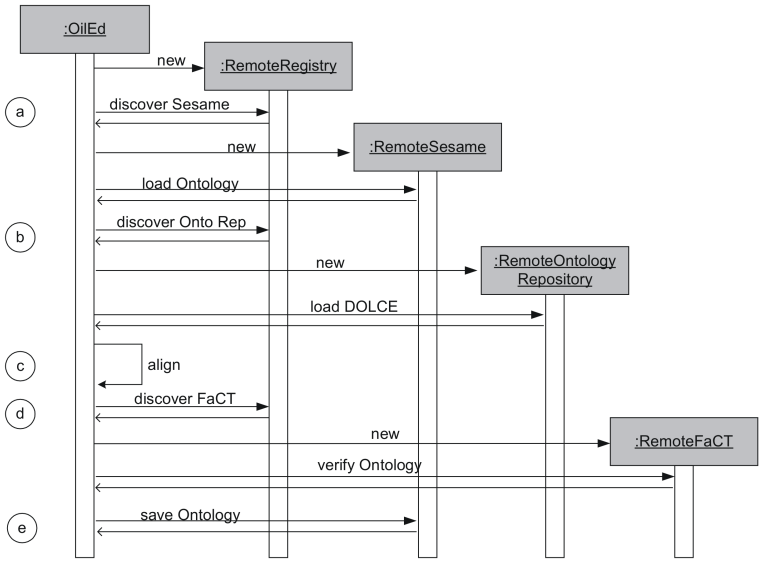

Fig. 6. OilEd's interaction with KAON SERVER

In a first step, OilEd creates a surrogate for the registry in order to query for a deployed Sesame (a). In our case, there exists only one Sesame component. However, if several would have been deployed, OilEd could specify additional criteria to refine its query (e.g. quality criteria like performance or reliability). The registry returns the ID of the component which is passed to RemoteSesame's constructor. The client is now able to work with the deployed Sesame and loads the desired ontology. In (b), a similar interaction takes place with the Ontology Repository component. OilEd uses its surrogate to load a

DOLCE foundational ontology. After that, the user can start aligning the HR ontology to DOLCE in step (c). For checking the consistency of the resulting ontology, OilEd queries the registry for a FaCT inference engine, creates a corresponding surrogate and starts the verification process (d). Finally, the aligned and consistent ontology is stored back into Sesame (e).

To conclude, the interoperation of heterogeneous software modules in the Semantic Web can make use of the proposed exchange syntaxes such as RDF and established languages such as RDF/S and OWL. This allows a basic means of interoperation based on the exchange of files. However, with an infrastructure like the KAON SERVER, we lift interoperation to the next level, making life easier for the developer of comprehensive Semantic Web applications.

8 Conclusion and Discussion

In this paper, we described the experiences of a case study in ontological engineering in a typical semantic web scenario, where ontological knowledge is not carefully designed from scratch but semi-automatically acquired from existing information sources. We described the use case and our experiences with applying a number of existing tools for handling ontological information.

Our experiences in the case study indicate that the approach of re-using and enriching structural knowledge from existing data sources as a basis for semantic web applications is viable. One of the questions addressed in this context is the question of how far this process can be automated and how much manual effort is needed to create useful ontologies. This question closely relates to a trade off between reusing and rebuilding ontologies. Discussions in the literature emphasize both the importance [28] and the danger of re-using existing knowledge [29].

On the one hand, ontologies are meant to be task neutral and capture generic structures of a domain and should therefore be reusable to a large degree. Considering the enormous effort of building large scale ontologies like UMLS or the Gene ontology (in both cases a large group of people is concerned just with maintaining and updating the model) re-use is really essential to build systems in acceptable time. In cases, where we do not have elaborated ontologies, the approach of supporting the process of building an ontology using automated extraction techniques provides reasonable benefits. Furthermore, our small scenario underemphasises the benefits of re-using, but these are likely to be bigger for bigger legacy datasources. Also, one is likely to get better results when starting from richer legacy descriptions (UML, ER) instead of simply the relational schema. We are optimistic about achieving even better results when carefully selecting the source to start with.

On the other hand, practice shows that most existing ontologies are still heavily biased towards the originally intended use. This is even more true for structural information we find on the web in terms of database or XML schemas. This is the reason why the re-use of legacy data essentially requires human intervention in various steps. Comparing the initially extracted ontology with the final result clearly underlines this claim. Our main conclusion here is that providing tools that support the complete evolution process from the extraction to the deployment of an ontology is essential. Diagnostic tools are especially important, e.g., for detecting errors, tracking changes, and evaluating their impact. In summary, we think that the hybrid approach for ontology engineering is a good choice for the semantic web because it is neither feasible to create ontologies from scratch for every new application nor to use the result of automatic extraction without human intervention.

We also discussed the topic of evolution management for evolving ontologies and of providing interoperability between tools used in the process. This general topics are of major interest for the development of the semantic web. Existing tools and methodologies can only be used successfully if we can make the tools interact.

The user must also be able to trace the development of the ontology, and judge the impact on particular applications of the changes made during various stages of the development process. In the reported case study, semantic web technology was used to ensure interoperability between the different tools used. On a syntactic level, RDF was used to encode and exchange models between different tools. On a structural level, OWL provided a standard language for encoding ontological knowledge. The actual interaction of tools was mediated by the KAON broker architecture that connects functional components on the basis of agreed interface definitions. Summarizing, what was provided in the case study is interoperability on a syntactic level based on prior agreement on the nature of data and the functional profiles of the involved tools. A natural next step towards an exploitation of semantic web technology for tool interoperability is to achieve interoperability on a semantic level.

A key question in the provision of semantic interoperability is the nature, scope and level of detail of service and implementation descriptions as well as the ability to acquire them in an efficient way. As general reasoning about pre and post conditions is known to be very hard, it is not realistic to assume that we can find a generic way of describing any kind of service in a uniform way and still provide matchmaking and integration functionality. First experiments with the KAON broker indicated that matchmaking in a particular domain of interest, in our case semantic web tools, can be facilitated by using an ontology of different types of tools and their characteristic properties. In order to build such an ontology we can use the structure of existing software repositories that already classify software entities according to certain properties. Concerning

the description of the actual implementation, it can be assumed that at least the static part of the implementation knowledge can be acquired from software design (in particular UML models) and documentation (for example JavaDoc). It is not very likely that functional integration can be performed automatically without prior knowledge of the implementation, but automatically extracting implementation details is a promising way to support developers in making tools accessible via broker architectures like KAON.

References

[1] S. Staab, H.-P. Schnurr, R. Studer, Y. Sure, Knowledge processes and ontologies, IEEE Intelligent Systems, Special Issue on Knowledge Management 16 (1).

[2] Y. Sure, R. Studer, On-To-Knowledge Methodology — final version, On-To-Knowledge deliverable 18, Institute AIFB, University of Karlsruhe (2002).

[3] M. Fernández-López, A. Gómez-Pérez, J. P. Sierra, A. P. Sierra, Building a chemical ontology using Methontology and the Ontology Design Environment, Intelligent Systems 14 (1).

[4] M. Uschold, M. King, Towards a methodology for building ontologies, in: Workshop on Basic Ontological Issues in Knowledge Sharing, held in conjunction with IJCAI-95, Montreal, Canada, 1995.

[5] T. R. Gruber, Towards Principles for the Design of Ontologies Used for Knowledge Sharing, in: N. Guarino, R. Poli (Eds.), Formal Ontology in Conceptual Analysis and Knowledge Representation, Kluwer Academic Publishers, Deventer, The Netherlands, 1993.
URL `citeseer.nj.nec.com/gruber93toward.html`

[6] A. Gomez-Perez, Ontology evaluation, in: S. Staab, R. Studer (Eds.), Handbook on Ontologies, Series on Handbooks in Information Systems, Springer Verlag, 2003.

[7] S. Dill, N. Eiron, D. Gibson, D. Gruhl, R. Guha, A. Jhingran, T. Kanungo, S. Rajagopalan, A. Tomkins, J. A. Tomlin, J. Y. Zien, Semtag and seeker: Bootstrapping the semantic web via automated semantic annotation, in: Proceedings of WWW 2003, 2003.
URL `http://www.tomkinshome.com/papers/2Web/semtag.pdf`

[8] H. Thompson, D. Beech, M. Maloney, N. Mendelsohn, Xml schema part 1: Structures, Recommendation, W3C, http://www.w3.org/TR/xmlschema-1/ (May 2001).

[9] Baclawski, Kokar, Kogut, Hart, Smith, Holmes, Letkowski, Aronson, Extending umlto support ontology engineering for the semantic web., in: Proc. of 4th Int. Conf. on UML - UML2001, 2001.

[10] J. van Griethuysen, Information processing systems – concepts and terminology for the conceptual schema and the information base, Tech. Rep. ISO/TR 9007:1987, ISO JTC 1/SC 32, International Standards Organization (ISO) (1987).

[11] D. Brickley, R. Guha., Rdf vocabulary description language 1.0: Rdf schema, W3c working draft, World Wide Web Consortium (April 2002).

[12] M. Dean, G. Schreiber, Owl web ontology language reference, W3c candidate recommendation, World Wide Web Consortium (August 2003).

[13] S. Abiteboul, R. Hull, V. Vianu, Foundation of databases, Addison-Wesley Publishing Company, 1995.

[14] C. Date, Referential integrity, in: Proceedings of Intl. Conf. on Very Large Data Bases (VLDB), 1981, pp. 2–12.

[15] L. Stojanovic, N. Stojanovic, R. Volz, Migrating data-intensive web sites into the semantic web, in: Proceedings of the 17th ACM symposium on applied computing (SAC), ACM Press, 2002, pp. 1100–1107.

[16] P. F. Patel-Schneider, P. Hayes, I. Horrocks, F. van Harmelen, Web ontology language (owl) abstract syntax and semantics, Working draft, W3C (November 2002).

[17] A. Gangemi, D. Pisanelli, G. Steve, An overview of the onions project: Applying ontologies to the integration of medical terminologies, Data Knowledge Engineering 31 (2) (1999) 183–220.

[18] C. Masolo, S. Borgo, A. Gangemi, N. Guarino, A. Oltramari, L. Schneider, The wonderweb library of foundational ontologies and the dolce ontology, WonderWeb Deliverable D17 - Preliminary Report (ver. 2.0, 15-08-2002) (2002).

[19] J. Z. Pan, I. Horrocks, Extending Datatype Support in Web Ontology Reasoning, in: Proc. of the 2002 Int. Conference on Ontologies, Databases and Applications of SEmantics (ODBASE 2002), 2002.
URL `download/2002/Pan-Horrocks-datatype-2002.pdf`

[20] S. Bechhofer, I. Horrocks, C. Goble, R. Stevens, OilEd: a Reason-able Ontology Editor for the Semantic Web, in: Proceedings of KI2001, Joint German/Austrian conference on Artificial Intelligence, Vol. 2174 of Lecture Notes in Artificual Intelligence, Springer-Verlage, Vienna, 2001, pp. 396–408.

[21] I. Horrocks, FaCT and iFaCT, in: P. Lambrix, A. Borgida, M. Lenzerini, R. Möller, P. Patel-Schneider (Eds.), Proceedings of the International Workshop on Description Logics (DL'99), 1999, pp. 133–135.
URL `download/1999/dl99-FaCT.ps.gz`

[22] S. Bechhofer, R. Möller, P. Crowther, The DIG Description Logic Interface, Submitted to DL'03 International Workshop on Description Logics (2003).

[23] M. Klein, D. Fensel, A. Kiryakov, D. Ognyanov, Ontology versioning and change detection on the web, in: 13th International Conference on Knowledge Engineering and Knowledge Management (EKAW02), no. 2473 in LNCS, Sigüenza, Spain, 2002, p. 197 ff.

[24] O. Lassila, R. R. Swick, Resource Description Framework (RDF): Model and Syntax Specification, Recommendation, World Wide Web Consortium, see http://www.w3.org/TR/REC-rdf-syntax/ (Feb. 1999).
URL `http://www.w3.org/TR/REC-rdf-syntax/`

[25] R. Volz, D. Oberle, S. Staab, B. Motik, KAON SERVER - a Semantic Web Management System, in: Proceedings of the Twelfth International World Wide Web Conference WWW12, Alternate Tracks, Practice and Experience, 20-24 May 2003, Budapest, Hungary, 2003.

[26] D. Oberle, R. Volz, B. Motik, S. Staab, An extensible open software environment, International Handbooks on Information Systems, Springer, 2003.

[27] H. C. Wong, K. Sycara, A taxonomy of middle-agents for the internet, in: Proceedings of the Fourth International Conference on MultiAgent Systems, 2000, pp. 465 – 466.

[28] A. Gómez-Pérez, Knowledge sharing and reuse, in: Liebowitz (Ed.), The handbook on Applied Expert Systems, CRC Press, 1998.

[29] A. Valente, T. Russ, R. MacGrecor, W. Swartout, Building and (re)using an ontology for air campaign planning, IEEE Intelligent Systems 14 (1) (1999) 27–36.