7 Applications of Metadata and Ontologies

A key value proposition enabled by the use of metadata descriptios based on concepts from domain specific ontologies, is the ability to describe Web and other types of content using semantic descriptions with fine grained abstractions. These descriptions could appear in the form of annotations in the case of unstructured data. Alternatively, in the case of structured data created according to a well defined schema, these descriptions can be created based on a mapping between the schema and a domain specific ontology. These metadata descriptions may also be used to query the underlying structured data as well. Finally, with the mappings between the schema and domain ontologies form a critical component, that enables domain ontolog driven information integration. In this chapter, we discuss:

- Structured and semi-structured metadata annotations of unstructured and semi-structured documents on the Web. Tools and techniques to support metadata annotation are presented in Section 7.1.
- Structured metadata annotations of structured Web resources with a well-defined set of types and schemas. Techniques to support schema and ontology mapping are discussed in Section 7.2.
- Approaches for ontology driven information integration are discussed in Section 7.3.

7.1 Tools and Techniques for Metadata Annotation

Knowledge about documents has traditionally been managed through the use of metadata, which can concern the world around the document, e.g., the author, and often at least part of the content, e.g., keywords. The Semantic Web proposes annotating document content using semantic information from domain ontologies. The result is Web pages with machine interpretable markup that provide the source material with which agents and Semantic Web Services can operate. The goal is to create annotations with well-defined semantics, however those semantics may be defined. This is a crucial requirement for interoperability, as it ensures that the annotator and annotation consumer actually share meaning.

Semantic Web annotations go beyond familiar textual annotations about the content of the documents, such as "clause seven of this contract has been deleted because . . ." and "the test results need to go in here". This kind of informal annotation is common in word processor applications and is intended primarily for use by

document creators. Semantic annotation formally identifies concepts and relations between concepts in documents, and is intended primarily for use by machines. For example, a semantic annotation might relate "Paris" in a text to an ontology which both identifies it as the abstract concept "City" and links it to the instance "France" of the abstract concept "Country", thus removing any ambiguity about which "Paris" it refers to. Annotations can be utilized to make the knowledge contained in unstructured sources (medical images such as X-rays) available in a structured form, allowing both accurate and focussed retrieval and knowledge sharing for a given patient's case. Moreover, they can be processed to automatically draft textual reports about the patient, the diagnostic information that is available and the assessments made about the data by the medical team.

In this section, we discuss some requirements identified for semantic annotation and review the systems that currently exist to support annotations of documents presented in [171]. Seven requirements, which are used to assess the capabilities of existing annotation systems, are identified for semantic annotation systems.

7.1.1 Requirements for Metadata Annotation

The metadata task may be considered from four viewpoints: ontologies, documents, annotations that link ontologies to documents, and end users. Each viewpoint suggests one or more requirements, e.g., the need for tools to support multiple, evolving ontologies (ontology viewpoint) and the need to support the reuse and versioning of documents (document viewpoint). Some requirements for metadata annotation are as follows [171]:

1. **Standard Formats**: Using standardized formats and data models is preferable whenever possible. Two types of standard are required, standards for describing ontologies such as OWL [45] and standards for annotations such as tRDF [42].

2. **User-Centered/Collaborative Design:** Since few organizations have the capacity to employ professional annotators, it is crucial to provide knowledge workers with easy to use interfaces that simplify the annotation process and place it in the context of their everyday work. Thre is a need to facilitate collaboration between users, with experts from different fields contributing to and reusing metadata annotations. Other issues for collaboration include implementing systems with access control functionality. For example, in a medical context, physicians might share all information about patients among themselves but only share anonymized information with planners. Issues related to access policies, trust and provenance are important in this context.

3. **Ontology Support (Multiple Ontologies and Evolution):** Metadata annotation tools need to be able to support multiple ontologies. For example, in a medical context, there may be one ontology for general metadata about a patient and other domain-specific ontologies that deal with diagnosis and medications. In addition, systems will have to cope with changes made to ontologies over time, such as incorporating new classes or modifying existing ones. This is a

crucial requirement as in some domains such as the biomedical domain, standardized vocabularies and ontologies are regularly updated. In this case, the problem is ensuring consistency between ontologies and annotations with respect to ontology changes. Some important issues for the design of an annotation environment are to determine how changes should be reflected in the knowledge base of annotated documents and whether changes to ontologies create conflicts with existing annotations. Knowledge workers may require facilities to help them explore and edit the ontologies they are using.

4. **Document Evolution (Document and Annotation Consistency):** Ontologies change sometimes but some documents change many times. What should happen to the annotations on a document when it is revised? Is it even desirable, in general, to transfer annotations to a new version of a document, or do versions of annotations need to be maintained in parallel with document versions. For example, if a contract were prepared for a new client, annotations that referred to a legal ontology could be retained, but annotations which referred to previous clients could be removed. How can this selective transfer of annotations be achieved?

5. **Annotation Storage:** The Semantic Web model assumes that annotations will be stored separately from the original document, whereas the "word processor" model assumes that comments are stored as an integral part of the document, which can be viewed or not as the reader prefers. The Semantic Web model, which decouples content and semantics, works particularly well for the Web environment in which the authors of annotations do not necessarily have any control over the documents they are annotating.

6. **Automation:** Easing the knowledge acquisition bottleneck can be enabled by the provision of facilities for automatic markup of document collections to facilitate the economical annotation of large document collections. To achieve this, the integration of knowledge extraction and natural language processing technologies into the annotation environment is vital.

7.1.2 Tools and Technologies for Metadata Annotation

In this section, we discuss annotation frameworks, tools and environments that produce semantic metadata annotations, i.e., metadata annotations that are based on a vocabulary presented by ontologies.

Metadata Annotation Frameworks

We discuss two frameworks for annotation in the Semantic Web, the W3C annotation project Annotea [172], and CREAM [173], an annotation framework being developed at the University of Karlsruhe. These frameworks can be implemented by multiple tools.

Annotea is a W3C project, which specifies infrastructure for annotation of Web documents, with emphasis on the collaborative use of annotations. The main format for Annotea is RDF and the kinds of documents that can be annotated are limited to HTML or XML-based documents. XPointer is used as the method for locating annotations within a document. The Annotea approach concentrates on a semiformal style of annotation, in which annotations are free text statements about documents. These statements must have metadata (author, creation time, etc.) and may be typed according to user-defined RDF schemas of arbitrary complexity. The storage model proposed is a mixed one with annotations being stored as RDF held either on local machines or on public RDF servers. The Annotea framework has been instantiated in a number of tools including Amaya, Annozilla and Vannotea, which are discussed later in this section.

The CREAM framework looks at the context in which annotations could be made. It specifies components required by an annotation system, including the annotation interface, with automatic support for annotators, the document management system and the annotation inference server. CREAM subscribes to W3C standard formats with annotations made in RDF or OWL and XPointers used to locate annotations in text, which restricts it to Web-native formats such as XML and HTML. The CREAM framework supports annotating the databases from which deep Web pages are generated so that the annotations are generated automatically with the pages. It is supported by a storage model that allows users to choose whether they want to store annotations separately on a server or embedded in a Web page. The CREAM framework allows for relational metadata, defined as "annotations which contain relationship instances". Relational metadata is essential for constructing knowledge bases which can be used to provide semantic services. Examples of tools based on the CREAM framework are S-CREAM and OntoMat-Annotizer, discussed later in the section.

Metadata Annotation Tools

The most basic annotation tools allow users to manually create annotations. They have a great deal in common with purely textual annotation tools but provide some support for ontologies. The W3C Web browser and editor Amaya [174] can mark-up Web documents in XML or HTML. The user can make annotations in the same tool they use for browsing and for editing text, making Amaya a good example of a single point of access environment. The Annozilla [179] browser aims to make all Amaya annotations readable in the Mozilla browser and to shadow Amaya developments. Teknowledge [180] produces a similar plug-in for Internet Explorer.

The Mangrove system is another example of manual but user-friendly annotation [175]. The annotation tool itself is a straightforward graphical user interface that allows users to associate a selection of tags with text that they highlight. Mangrove has recently been integrated with a semantic email service [176], which supports the initiation of semantic email processes, such as meeting scheduling, via text forms. The COHSE Annotator [188] produces annotations that are compatible

with Annotea. The annotator is provided as a plug-in suitable for use in Mozilla or Internet Explorer, giving the user a choice of working environment. The COHSE architecture has been used to support a number of domain applications, including the generation of semantic annotation for visually impaired users [190] and enriching a Java tutorial site [189].

Multimedia annotation is the next phase of development for annotation, expanding the range files types that can be marked up into images, video and audio. Vannotea [177] has been developed by the University of Brisbane for adding metadata to MPEG-2 (video), JPEG 2000 (image) and Direct 3D (mesh) files, with the mesh being used to define regions of images. It has been designed to allow input from distributed users enabling deployment to annotate cultural artifacts in a collaborative annotation exercise involving both museum curators and indigenous groups [177].

Some manual annotation tools provide more sophisticated user support and a degree of semi-automatic or automatic annotation facilities. The OntoMat-Annotizer is a tool for making annotations based on the CREAM framework. A Web browser displays the page being annotated and provides user-friendly function, such as drag-and-drop creation of instances and the ability to markup pages while they are being created. OntoMat has been extended to include support for semi-automatic annotation. The first of these extensions was S-CREAM [181], which uses an information extraction (IE) system (Amilcare [182]). The system learns how to reproduce the user annotation, to be able to suggest annotations for new documents. OntoMat also incorporates methods for deep annotation [183]. M-OntoMat-Annotizer [184] supports manual annotation of image and video data by indexers with little multimedia experience by automatic extraction of low-level features that describe objects in the content.

SHOE Knowledge Annotator [186] was an early system which allowed users to markup HTML pages in SHOE guided by ontologies available locally or via a URL. Users were assisted by being prompted for inputs. Running SHOE took a step toward automated markup by assisting users to build wrappers for Web pages that specify how to extract entities from lists and other pages with regular formats. A recent addition is the RDF annotator SMORE [185] which allows mark-up of images and emails as well as HTML and text. A tool with similar characteristics to SMORE is the Open Ontology Forge (OOF) [187], an ontology editor that supports annotation, taking it a step further toward an integrated environment to handle documents, ontologies and annotations.

Automation can generally be regarded as falling into three categories. The most basic kind uses rules or wrappers written by hand that try to capture known patterns for the annotations. Supervised systems learn from sample annotations marked up by the user. A problem with these methods is that picking enough good examples is a nontrivial and error-prone task. In order to tackle this problem unsupervised systems employ a variety of strategies to learn how to annotate without user supervision, but their accuracy is limited.

Lixto is a Web information extraction system which allows wrappers to be defined for converting unstructured resources into structured ones. The tool allows users to create wrappers interactively and visually by selecting relevant pieces of information [191]. MnM was designed to markup training data for IE tools rather than as an annotation tool per se [192]. It stores marked up documents as tagged versions of the original, rather than in RDF format. It provides an HTML browser to display the document and ontology browser features. MnM provides open APIs to link to ontology servers and for integrating information extraction tools, making it flexible with the formats and methods it uses.

Melita [193] is a user-driven automated semantic annotation tool which makes two main strategies available to the user. It provides an underlying adaptive information extraction system (Amilcare) that learns how to annotate the documents by generalizing on the user annotations. It also provides facilities for rule writing (based on regular expressions) to allow sophisticated users to define their own rules. Documents are not selected based on the expected usefulness, to the IE system, of annotating the document. The Amilcare IE system has been incorporated in K@, a legal KM system with RDF-based semantic capabilities produced by Quinary [194].

CAFETIERE is a rule-based system for generating XML annotations developed as part of the Parmenides project [195]; it has been used to annotate the GENIA biomedical corpus [208]. Text mining techniques supplemented with slot-based constraints are used to suggest annotations to analysts [196]. The Parmenides project also experimented with a clustering approach to suggest concepts and relations to extend ontologies [197].

Armadillo is a system for unsupervised creation of knowledge bases from large repositories (e.g., the Web) as well as for document annotation [198]. It uses the redundancy of the information in repositories to bootstrap learning from a handful of seed examples selected by the user. Seeds are searched in the repository. Then Adaptive IE is used to generalize over the examples and find new facts. Confirmation by several sources (e.g., documents) is then required to check the quality of the newly acquired data. After confirmation, a new round of learning can be initiated. This bootstrapping process can be repeated until the user is satisfied with the quality of the learned information.

KnowItAll [199] automates extraction of large knowledge bases of facts from the Web. The pointwise mutual information (PMI) measure is used. The PMI measure is roughly the ratio between the number of search engine hits obtained by querying with the discriminator phrase (e.g., "Liege is a city") and the number of hits obtained by querying with the extracted fact (e.g., "Liege"). Three extensions to the system (pattern learning, subclass extraction and list extraction) which are shown to improve overall performance have also been provided.

The SmartWeb project is also investigating unsupervised approaches for RDF knowledge base population [200]. Their approach uses class and subclass names from the ontology to construct examples. The context of these examples is then learned. In this way, instances can be identified which have similar contexts, but

which may use different terminology from the ontology. SmartWeb is aimed at broadband mobile access.

Another approach to learning annotations which exploits the sheer size of the Web is Pattern-based Annotation through Knowledge On the Web (PANKOW) [201]. PANKOW uses a range of relatively rare, but informative, syntactic patterns to markup candidate phrases in Web pages without having to manually produce an initial set of marked-up Web pages and go through a supervised learning step. AeroSWARM8 is an automatic tool for annotation using OWL ontologies based on the DAML annotator AeroDAML [202]. This has both a client/server version and a Web-enabled demonstrator in which the user enters a URI and the system automatically returns a file of annotations on another Web page.

SemTag is another example of a tool which focusses only on automatic markup [124]. It is based on IBM's text analysis platform Seeker and uses similarity functions to recognize entities which occur in contexts similar to marked-up examples. The key problem of large-scale automatic markup is identified as ambiguity, e.g., identical strings, such as "Niger", which can refer to different things, a river or a country. A Taxonomy-Based Disambiguation (TBD) algorithm is proposed to tackle this problem. SemTag is proposed as a bootstrapping solution to get a semantically tagged collection off the ground. It is intended as a tool for specialists rather than one for knowledge workers.

KIM [203] [204] uses information extraction techniques to build a large knowledge base of annotations. The annotations in KIM are metadata in the form of named entities (people, places, etc.) which are defined in the KIMO ontology and identified mainly from references to extremely large gazetteers. In the Rich News application KIM has been used to help annotate television and radio news by exploiting the fact that Web news stories on the same topic are often published in parallel [207].

The Rainbow project is taking a Web-mining-led approach to automating annotation. Rainbow is in fact a family of independent applications which share a common Webservice front end and upper-level ontology [205]. The applications include text mining from product catalogs as well as more general pattern-matching applications such as pornography recognition in bit map image files. The generated RDF is stored in Sesame databases for semantic retrieval [210].

A traditional approach to information extraction is used by the h-TechSight Knowledge Management Platform, in which the GATE rule-based IE system is used to feed a semantic portal [206]. This work is of particular interest because the automatically generated annotations are monitored to produce metrics describing the "dynamics" of concepts and instances which can be fed back to end users [209]. It is envisaged that dynamics data will be used to inform the manual evolution of ontologies.

Integrated Annotation Environments

In this section, we discuss systems that are aimed at integrating annotation into standard tools and making annotation simultaneous to writing. WiCKOffice [211] demonstrates how writing within a knowledge-aware environment has useful support possibilities, such as automatic assistance for form filling using data extracted from knowledge bases. AktiveDoc [212] enables annotation of documents at three levels: ontology-based content annotation, free text statements and on-demand document enrichment. Support is provided during both editing and reading. Semi-automatic annotation of content is provided via adaptive information extraction from text (using Amilcare). AktiveDoc is designed for knowledge reuse; it is able to monitor editing actions and to provide automatic suggestions about relevant content. Armadillo supports searches of relevant knowledge in large repositories; annotations in the document are used as context for searches. Annotations are saved in a separate database; levels of confidentiality are associated with annotations to ensure confidentiality of knowledge when necessary. AeroDAML can provide automation within authoring environments. For example, the SemanticWord annotator [213] provides graphical-user-interface based tools to help analysts annotate Microsoft Word documents with DAML ontologies as they write.

Two systems discussed next, are not strictly annotation tools, but produce annotation-like services on demand for users browsing unannotated resources. Magpie [214] operates from within a Web browser and does "real-time" annotation of Web resources by highlighting text strings related to an ontology of the user's choice. The Thresher uses wrappers to generate RDF on the fly as users browse deep Web resources [215]. The user can access semantic services for recognized objects. Writing wrappers is a complex task which Thresher tackles by providing facilities for nontechnical users to markup examples of a particular class. These are then used to induce wrappers automatically. Thresher is part of the Haystack semantic browser [216], which enables users to personalize the ontologies they use.

7.1.3 Comparative Evaluation

We now revisit the requirements presented in Section 7.1.1, and discuss a comparative evaluation of the various tool discussed in the previous section with respect to the requirements.

1. **Standard Formats:** The discussion in the previous section shows that the W3C standards, particularly Annotea, are becoming dominant in this area. Systems like CAFETIERE, which use their own XML-based annotation scheme, are rare. This requirement has been fulfilled, although the standards may need to be augmented to tackle inadequacies in the existing standards.

2. **User-Centered/Collaborative Design:** The most common home environment of the tools we have seen is a Web browser, a natural result of the fact that most of them were designed for the Semantic Web. The downside is that it both

focusses development on native Web formats like HTML and XML and tends to divorce the annotation process from the process of document creation. More attention needs to be paid to developing built-in or plug-in semantic annotation facilities in commonly used packages to encourage knowledge workers to view annotation as part of the authoring process, not as an afterthought, and also to support annotation in collaborative environments, as for example in Vannotea. Most of the tools discussed in the previous section did not address issues of provenance or access rights. Standard methods to restrict access to databases or the file system are available. As a result of offering this kind of support for trust, provenance and access policies concerning annotations are important issues which need to be addressed.

3. **Ontology Support (Multiple Ontologies and Evolution):** Annotation tools have adapted rapidly to recent changes in ontology standards for the Web, with many of the more recent tools already supporting OWL. However, support for doing anything more complex than searching and navigating an ontology browser is the exception. Ontology maintenance, which directly affects the maintenance of annotations, is poorly supported, or not supported at all, by the current generation of tools. This perhaps reflects the assumption that knowledge workers will use existing ontologies rather than editing or creating them. However there are signs that annotation systems are giving users more control of ontologies. Melita allows users to split a concept and then view all the instances that have been created for the old concept and reassign them. The COHSE architecture includes a component for maintaining the ontology but this does not appear to be available from the annotator. The Open Ontology Forge supports the creation of new classes from a root class. h-TechSight monitors the dynamics of instances and concepts to assist endusers in manual ontology evolution. Parmenides has gone further and experimented with clustering methods to suggest ontology changes. However there is still a long way to go and we believe that ontology maintenance presents a significant research challenge.

4. **Document Evolution (Document and Annotation Consistency):** Keeping annotations synchronized with changes to documents is challenging and this is one area in which the current annotation standards are inadequate. The Annotea approach adopted by many of the tools stores annotations separately from the document and uses XPointer to locate them in the document. There are strong arguments in favor of separate storage of annotations and documents, but the problem with the XPointer approach is that connections are one-way from annotations to documents and, therefore, too easily broken by edits at the document end. An environment in which documents and annotations are stored separately, but closely coordinated is required. A number of practical fixes have been implemented in OntoMat, including the ability to search for similar documents that have already been annotated, and a proposal to use pattern matching to help relocate annotations in suitable places in the new document. However, a coordinated approach is needed to tackle the issues of versioning annotations as

documents evolve. These include determining who has permission to edit annotations, at which points in the document life cycle is it appropriate to update the annotations, and what automatic interventions are possible to reduce the burden on users.

5. **Annotation Storage:** In the Semantic Web, documents and their annotations are stored separately. This is unavoidable since documents and annotations are likely to be owned by different people or organizations and stored in different places. A variety of approaches to separate storage were seen in the tools discussed. The Annotea approach calls for RDF servers. Web storage technologies that have been used are RDF triplestore (Armadillo and AktiveDoc), Label Bureaus (SemTag) and DLS (COHSE). An alternative model is to store annotations directly in the document. This approach has been used for in Semantic-Word and MnM. Separate storage of annotations results in decoupling of semantics and content and facilitates document reuse because it is possible to set up rules which control and automate which kinds of annotations are transferred to new documents and which are not. It allows information from heterogeneous resources to be queried centrally as a knowledge base. It also makes it easy to produce different views of a document for users with different roles in an organization or different access rights, thus facilitating knowledge sharing and collaboration. The results of the comparative evaluation of the various tools with respect to the above requirements is presented in Table 7.1 below.

Table 7.1. Comparison of metadata tools

Annotation Tool	User-Centered Design	Ontology support	Document Evolution	Annotation Storage
Amaya	Web browser, editor	Annotation server	XPointer	Local, annotation server
Mangrove	Graphical annotation tool			RDF database (Jena)
Vannotea	Collaboration support			Annotation server
OntoMat	Drag/drop, create, annotate	Ontobroker annotations inference server	Xpointer, pattern matching	Annotation server, embedded in Web page, separate file
M-OntoMat Annotizer	Extraction of visual descriptors			Annotation server
SHOE Knowledge Annotator	Prompting	Ontology server		Embedded in Web page
SMORE	Web browser, editor	Ontology server, editing		
Open Ontology Forge	Web browser, drag, drop, create, annotate	Local, editable ontologies	Xpointer	Local RDF or XML file
COHSE Annotator	Plug-in for Mozilla and Internet Explorer	Ontology server	Xpointer	Annotation server

Table 7.1. Comparison of metadata tools

Annotation Tool	User-Centered Design	Ontology support	Document Evolution	Annotation Storage
Lixto				
MnM	Web browser	Ontology server	Store anno-tated page	Embedded inWeb page
Melita	Control IE intrusive-ness	Local, editable ontologies	Regular expressions	
Parmenides		Additions based on clustering		
Armadillo				RDF triple store
KnowItAll				
SmartWeb				RDF Knowledge base
PANKOW	CREAM			
Aero-SWARM	Web Services	Local ontologies		
SemTag				Label Bureau (PICS)
KIM	Various plug-in front ends	KIMO		RDF Knowledge-base
Rainbow Project	AmphorA XHTML database	Shared upper-level ontology		RDF repository (Sesame)
h-TechSight	KM Portal	Ontology editor, dynamics metrics		Tagged HTML web server
WiCKOf-fice	Office application, support for form fill-ing			Annotation server
AktivDoc	Integrated editing environment			RDF triple store
Semantic-Word	Microsoft Word GUIs		Markup tied to text regions	
Magpie	Web browser plug-in			
Thresher	Haystack semantic browser	Ontology person-alization		

6. **Automation:** Automation is vital to ease the knowledge acquisition bottleneck, as discussed above. Many of the systems we examined had some kind of auto-matic and semi-automatic support for annotation. Most of these handled just text, using mainly wrappers, IE and natural language processing although there are some systems, notably M-OntoMat-Annotizer and parts of the Rainbow Project, looking to automate the handling of other media. Language technolo-

gies present usability challenges when deployed for knowledge workers since most are research tools or designed for use by specialists. A first step in addressing these challenges is Melita, where attention has been paid in finding ways to enable a seamless user interaction with the underlying IE system. In addition to the usability challenges there are also research challenges, among which extraction of relations is important for semantic annotation. A comparison of annotation tools for automation is presented in Table 7.2 below.

Table 7.2. A comparison of annotation tools based on automation support

Annotation Tool	Automation	Type of Analysis	Learning
Amaya	No		
Mangrove	No		
Vannotea	No		
OntoMat	Yes	PANKOW, Amilcare (IE)	Supervised learning
M-OntoMat Annotizer	Yes	Extraction of spatial description	Genetic algorithms
SHOE Knowledge Annotator	Yes	Running SHOE (wrappers)	No
SMORE	Yes	Screen scraper	No
Open Ontology Forge	Yes	String matching	No
COHSE Annotator	Yes	Ontology string matching	No
Lixto	Yes	Wrappers	No
MnM	Yes	POS tagging, named entity recognition	Supervised learning
Melita	Yes	String matching, POS tagging, named entity recognition	Supervised learning
Parmenides	Yes	Text mining with constraints	Unsupervised learning
Armadillo	Yes	String matching, POS tagging, named entity recognition	Unsupervised learning
KnowItAll	Yes	String matching, Hearst patterns	Unsupervised learning
SmartWeb	Yes	Shallow linguistic parsing	Unsupervised learning
PANKOW	Yes	Hearst patterns	Unsupervised learning
AeroSWARM	Yes	AeroText	No
SemTag	Yes	Seeker, similarity, TBD	Unsupervised learning
KIM	Yes	String matching, POS tagging. named entity recognition	No

Table 7.2. A comparison of annotation tools based on automation support

Annotation Tool	Automation	Type of Analysis	Learning
Rainbow Project	Yes	Hidden markov models, bit-map classification	Supervised learning
h-TechSight	Yes	Shallow linguistic analysis	No
WiCKOffice	Yes	Named entity recognition	No
AktivDoc	Yes	String matching, POS tagging, named entity recognition	Unsupervised and supervised learning
SemanticWord	Yes	AeroDAML	No
Magpie	Yes	String matching, named entity recognition	No
Thresher	Yes	Screen scraping, wrappers	Supervised learning

7.2 Techniques for Schema/Ontology Mapping

Schema and ontology matching is a critical problem in many application domains, such as Semantic Web, schema/ontology integration, data warehouses, and e-commerce. Many different matching solutions have been proposed so far. In the following we present a discussion of schema and ontology matching techniques based on classifications presented in [217] [218].

7.2.1 A Classification of Schema-matching Approaches

Schema-matching approaches can be classified as follows [217] [218]:

- **Elementary matchers:** These consist of instance-based and schema-based, element- and structure-level, linguistic- and constraint-based matching techniques.
- **Combination of matchers:** These consist of various ways of combining the schema matchers using committee-based or hybrid approaches.

Elementary schema-based matching techniques are classified based on two perspectives (Figure 7.1). These two perspectives are presented as two trees sharing their leaves. The leaves represent classes of elementary matching techniques and their concrete examples, identified as basic techniques in the figure. The two perspectives are discussed below.

Granularity/Input Interpretation: This is based on the granularity of the match, i.e., whether it is at the element or structural level, and how these techniques interpret this information. This perspective is illustrated from the top in a descending manner in Figure 7.1 till it reaches the Basic Techniques Layer. Elementary matchers are further distinguished based on the following criteria:

Element-level vs. structure-level. Element-level matching techniques compute mapping elements by analyzing entities in isolation, ignoring their relations with

other entities. Structure-level techniques compute mapping elements by analyzing how entities appear together in a structure.

Syntactic vs. external vs. semantic. The key characteristic of the syntactic techniques is that they interpret the input as a function of its syntactic structure. External techniques exploit auxiliary (external) resources of a domain and common knowledge in order to interpret the input. These resources might be human input or some thesaurus expressing the relationships between terms. The key characteristic of the semantic techniques is that they use some formal semantics (e.g., model-theoretic semantics), possibly with some sort of reasoning to interpret the input and justify their results.
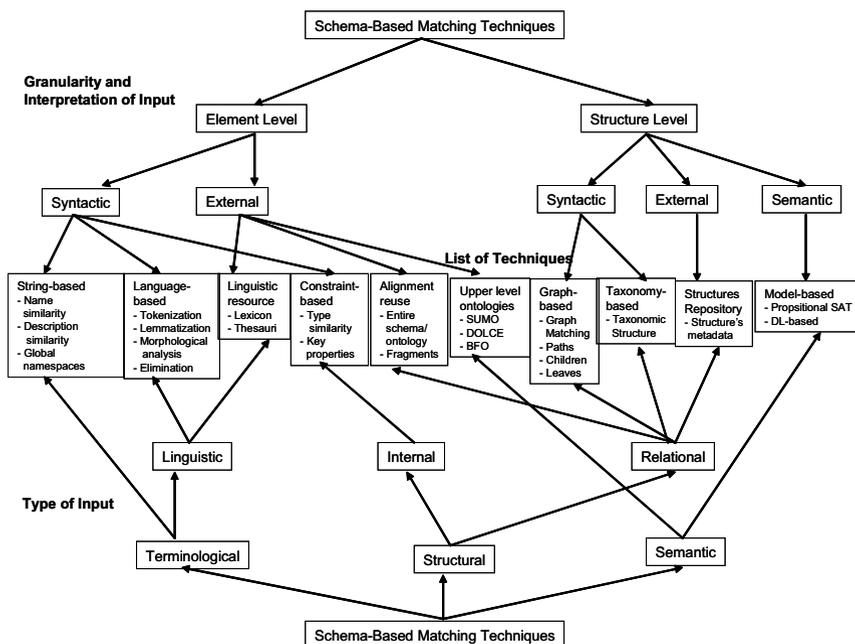

Fig. 7.1. A Classification of schema-based matching approaches

Type of Input. This is based on the type of input used by the elementary matching techniques. This perspective is illustrated from the bottom in an ascending manner in Figure 7.1 till it reaches the Basic Techniques Layer. Elementary matchers are further distinguished based on the following criteria:

- The first level is categorized depending on which kind of data the algorithms work on: string (*terminological*), structure (*structural*) or model (*semantics*). The two first ones are found in the ontology descriptions, the last one requires some semantic interpretation of the ontology and usually uses some semantically-compliant reasoner to deduce the correspondences.

- The second level of this classification decomposes further these categories if necessary: *terminological* methods can be *string-based* (considering the terms as sequences of characters) or based on the interpretation of these terms as linguistic objects (*linguistic*). The structural methods category is split into two types of methods: those which consider the *internal* structure of entities (e.g., attributes and their types) and those which consider the relation of entities with other entities (*relational*).

We discuss below the main classes of the Basic Techniques Layer and the associated matching systems according to the above classification in more detail. Techniques based on upper-level ontologies and DL-based techniques have not been implemented in any matching system yet. However, their use in matching systems seems quite likely in the near future.

Element-level techniques

String-based techniques consider strings as sequences of letters in an alphabet. They assume that the more similar the strings, the more likely they denote the same concepts. A comparison of different string-matching techniques, from distance-like functions to token-based distance functions can be found in [219]. Some examples of string-based techniques which are extensively used in matching systems are *prefix/suffix*, *edit distance*, and *n-gram*.

Prefix/Suffix. Two strings are input and a check of whether the first string starts/ends with the second one is performed. Prefix is efficient in matching cognate strings and similar acronyms (e.g., int and integer). This test can be transformed into a smoother distance by measuring the relative size of the prefix and the strings. These techniques have been used in [225] [231] [232] [233].

Edit distance. This distance takes as input two strings and computes the edit distance between the strings, that is, the number of insertions, deletions, and substitutions of characters required to transform one string into another, normalized by the length of the longest string.

N-gram. This test takes as input two strings and computes the number of common n-grams (i.e., sequences of n characters) between them. These techniques have been used in [225] [231] [234].

Language-based techniques consider names as words in some natural language (e.g., English) and apply Natural Language Processing (NLP) techniques that exploit morphological properties of the input words.

Tokenization. Names of entities are parsed into sequences of tokens by a tokenizer which recognizes punctuation, cases, blank characters, digits, etc. (e.g., see [230]).

Lemmatization. The strings, underlying tokens are morphologically analyzed in order to find all their possible basic forms (e.g., see [230]).

Elimination. The tokens that are articles, prepositions, conjunctions, and so on, are marked to be discarded (e.g., see [232]).

Usually, the above-mentioned techniques are applied to names of entities before running string-based or lexicon-based techniques in order to improve their results. However, language-based techniques may be considered as a separate class of matching techniques, since they can be naturally extended, for example, in a distance computation (by comparing the resulting strings or sets of strings).

Constraint-based techniques are algorithms which deal with the internal constraints being applied to the definitions of entities, such as types, cardinality of attributes, and keys.

Datatype comparison involves comparing the various attributes of a class with regard to the datatypes of their value. Contrary to objects that require interpretation, the datatypes can be considered objectively and it is possible to determine how a datatype is close to another (ideally this can be based on the interpretation of datatypes as sets of values and the set-theoretic comparison of these datatypes). For instance, the datatype `day` can be considered closer to the datatype `workingday` than the datatype `integer`. This technique is used in [228].

Multiplicity comparison attribute values can be collected by a particular construction (set, list, multiset) on which cardinality constraints are applied. It is possible to compare the so constructed datatypes by comparing (i) the datatypes on which they are constructed and (ii) the cardinality constraints that are applied to them. For instance, a set of between two and three children is closer to a set of three people than a set of ten to twelve flowers (if children are people). This technique is used in [228].

Linguistic resources such as common knowledge or domain-specific thesauri are used to match words (in this case names of schema/ontology entities are considered as words of a natural language) based on linguistic relations between them (e.g., synonyms, hyponyms).

Common knowledge thesauri are used to obtain the meaning of terms used in schemas/ontologies. For example, WordNet [237] is an electronic lexical database for English (and other languages), where various senses (possible meanings) of words or expressions are put together into sets of synonyms. Relations between schema/ontology entities can be computed in terms of bindings between WordNet senses; see, for instance [221] [230]. Other matchers exploit thesauri based on their structural properties, e.g., WordNet hierarchies. In particular, hierarchy-based matchers measure the distance, for example, by counting the number of arcs traversed, between two concepts in a given hierarchy.

Domain-specific thesauri usually store some specific domain knowledge, which is not available in common knowledge thesauri (e.g., proper names) as entries with synonym, hypernym and other relations; see, for instance [232].

Alignment reuse techniques exploit alignments of previously matched schemas and ontologies, for instance, when we need to match schema/ontology o and o'', given the alignments between o and o', and between o' and o'' from the external resource, storing previous match operation results. The alignment reuse is motivated by the intuition that many schemas/ontologies to be matched are similar to already-matched schemas/ontologies, especially if they are describing the same

application domain. These techniques are particularly promising when dealing with large schemas/ontologies consisting of hundreds and thousands of entities. In these cases, first, large match problems are decomposed into smaller sub-problems, thus generating a set of schema/ontology fragment-matching problems. Then, reusing previous match results can be more effectively applied at the level of schema/ontology fragments compared to entire schemas/ontologies. The approach was first introduced in [217], and later was implemented as two matchers, i.e., reuse of (i) entire schemas/ontologies alignments, or (ii) their fragments; see, for details [220] [225] [235].

Upper-level formal ontologies can be also used as external sources of common knowledge. Examples are the Suggested Upper Merged Ontology (SUMO) [49] and Descriptive Ontology for Linguistic and Cognitive Engineering (DOLCE) [92]. The key characteristic of these ontologies is that they are logic-based systems, and therefore, matching techniques exploiting them can be based on the analysis of interpretations. Even though current matching systems do not use these techniques, it is likely that this will happen in the near future. In fact, the DOLCE ontology aims at providing a formal specification (axiomatic theory) for the top-level part of WordNet. Therefore, systems exploiting WordNet now in their matching process might also consider using DOLCE as a potential extension.

Structure-level techniques

Graph-based techniques view database schemas, taxonomies and ontologies as graph-like structures containing terms and their interrelationships. Usually, the similarity comparison between a pair of nodes from the two schemas/ontologies is based on the analysis of their positions within the graphs. The intuition behind this is that if two nodes from two schemas/ontologies are similar, their neighbors might also be somehow similar.

Graph matching. Matching graphs is a combinatorial problem and is usually solved by approximate methods. In schema/ontology matching, the problem is encoded as an optimization problem (finding the graph matching minimizing some distance like the dissimilarity between matched objects) which is further resolved with the help of a graph-matching algorithm. This optimization problem is solved through a fix-point algorithm (improving gradually an approximate solution until no improvement is made). Examples of such algorithms are [233] and [228].

Children. The (structural) similarity between inner nodes of the graphs is computed based on similarity of their children nodes, that is, two non-leaf schema elements are structurally similar if their immediate children sets are highly similar. A more complex version of this matcher is implemented in [225].

Leaves. The (structural) similarity between inner nodes of the graphs is computed based on similarity of leaf nodes, that is, two non-leaf schema elements are structurally similar if their leaf sets are highly similar, even if their immediate children are not; see, for example [225] [232].

Relations. The similarity computation between nodes can also be based on their relations. For example, if class `Photo` and `Camera` relates to class `NKN` by relation `hasBrand` in one ontology, and if class `DigitalCamera` relates to class `Nikon` by relation `hasMarque` in the other ontology, then knowing that classes `Photo` and `Camera` and `DigitalCamera` are similar, and also relations `hasBrand` and `hasMarque` are similar, we can infer that `NKN` and `Nikon` may be similar.

Taxonomy-based techniques consider only the specialization relation. The intuition behind taxonomic techniques is that is-a links connect terms that are already similar (each being a subset of the other); therefore their neighbors may be also somehow similar.

Bounded path matching. Bounded path matchers take two paths with links between classes defined by the hierarchical relations, compare terms and their positions along these paths, and identify similar terms; see, for instance [234].

Super(sub)-concept rules. These matchers are based on rules capturing the above stated intuition. For example, if super-concepts are the same, the actual concepts are similar to each other. If sub-concepts are the same, the compared concepts are also similar; see, for example [224] [226].

Repository of structures stores schemas/ontologies and their fragments together with pairwise similarities (e.g., coefficients in the [0,1] range) between them. When new structures are to be matched, they are first checked for similarity to the structures which are already available in the repository. The goal is to identify structures which are sufficiently similar to be worth matching in more detail, or to reuse already existing alignments. Obviously, the determination of similarity between structures should be computationally cheaper than matching them in full detail. In order to match two structures, [235] proposes using some metadata describing these structures, such as structure name, root name, number of nodes, maximal path length, etc. Then, these indicators are analyzed and are aggregated into a single coefficient, which estimates the similarity between them.

Model-based algorithms handle the input based on its semantic interpretation (e.g., model-theoretic semantics). Examples are propositional satisfiability (SAT) and description logics (DL) reasoning techniques. As from [221] [229] [230], the approach is to decompose the graph(tree)-matching problem into a set of node-matching problems. Then, each node-matching problem, namely each pair of nodes with possible relations between them, is translated into a propositional formula of form, *Axioms => rel(context$_1$, context$_2$)*, and checked for validity. Axioms encode background knowledge (e.g., `HypertrophicCardioMyopathy subClassOf Disease` codifies the fact that Hypertrophic Cardiomyopathy is a kind of disease), which is used as premises to reason about relations rel (e.g., `=`, `subClassOf`, `unsatisfiability`) holding between the nodes *context$_1$* and *context$_2$*. A propositional formula is valid iff its negation is unsatisfiable. The unsatisfiability is checked by using state-of-the-art SAT solvers. Propositional language used for codifying matching problems into propositional unsatisfiability problems is limited in its expressiveness; namely it allows for handling only unary predicates. Thus, it cannot handle, for example, binary predicates, such as properties or roles, which

are expressible in OWL and various variants of DLs. The relations (e.g., `=`, `sub-ClassOf`, `unsatisfiability`) can be expressed using subsumption in DLs. In fact, first merging two ontologies (after renaming) and then testing each pair of concepts and roles for subsumption is enough for aligning terms with the same interpretation (or with a subset of the interpretations of the others). Currently, there are no systems supporting DL-based techniques.

7.2.2 Schema-matching Techniques: Overview

We now look at some recent schema-based state-of-the-art matching systems in the context of the classification presented in Figure 7.1. A summary of the various characteristics of these techniques is presented in Table 7.3.

Similarity Flooding. The Similarity Flooding (SF) [233] approach utilizes a hybrid-matching algorithm based on the ideas of similarity propagation. Schemas are presented as directed labeled graphs; the algorithm manipulates them in an iterative fix-point computation to produce an alignment between the nodes of the input graphs. The technique starts with string-based comparison (common prefix and suffix tests) of the vertex labels to obtain an initial alignment which is refined within the fix-point computation. The basic concept behind the SF algorithm is the similarity spreading from similar nodes to the adjacent neighbors through propagation coefficients. From iteration to iteration the spreading depth and the similarity measure increase till the fix-point is reached. The result of this step is a refined alignment which is further filtered to finalize the matching process. SF considers the alignment as a solution to a clearly stated optimization problem.

Artemis. Analysis of Requirements: Tool Environment for Multiple Information Systems (Artemis) [222] was designed as a module of the MOMIS mediator system [238] for creating global views. It performs affinity-based analysis and hierarchical clustering of source schema elements. Affinity-based analysis represents the matching step: in a hybrid manner it calculates the name, structural and global affinity coefficients exploiting a common thesaurus. The common thesaurus is built with the help of Ontology Development Tools, WordNet or manual input. It represents a set of intensional and extensional relationships which depict intra- and inter-schema knowledge about classes and attributes of the input schemas. Based on global affinity coefficients, a hierarchical clustering technique categorizes classes into groups at different levels of affinity. For each cluster it creates a set of global attributes and the global class. The logical correspondence between the attributes of a global class and source schema attributes is determined through a mapping table.

Cupid. Cupid [232] implements a hybrid-matching algorithm comprising linguistic and structural schema-matching techniques, and computes similarity coefficients with the assistance of a domain-specific thesaurus. Input schemas are encoded as graphs. Nodes represent schema elements and are traversed in a combined bottom-up and top-down manner. The matching algorithm consists of three phases and operates only with tree structures to which non-tree cases are reduced.

The first phase (linguistic matching) computes linguistic similarity coefficients between schema element names (labels) based on morphological normalization, categorization, string-based techniques (common prefix, suffix tests) and a thesauri lookup. The second phase (structural matching) computes structural similarity coefficients weighted by leaves which measure the similarity between contexts in which elementary schema elements occur. The third phase (mapping elements generation) computes weighted similarity coefficients and generates final alignment by choosing pairs of schema elements with weighted similarity coefficients which are higher than a threshold.

COMA. COmbination ofMAtching algorithms (COMA) [225] is a composite schema-matching tool. It provides an extensible library of matching algorithms, a framework for combining obtained results, and a platform for the evaluation of the effectiveness of the different matchers. Matching library is extensible, and contains six elementary matchers, five hybrid matchers, and one reuse-oriented matcher. Most of the matchers implement string-based techniques (affix, n-gram, edit distance, etc.) as a background idea; others share techniques with Cupid (thesauri look-up, etc.); and the reuse-oriented matcher tries to reuse previously obtained results for entire new schemas or for its fragments. Schemas are internally encoded as DAGs, where the elements are the paths. This aims at capturing contexts in which the elements occur. Distinct features of the COMA tool with respect to Cupid are a more flexible architecture and a possibility of performing iterations in the matching process.

NOM. Naive Ontology Mapping (NOM) [227] adopts the idea of composite matching from COMA [225]. Some other innovations with respect to COMA are in the set of elementary matchers based on rules exploiting explicitly codified knowledge in ontologies, such as information about super- and sub-concepts and super- and sub-properties. At present the system supports 17 rules. For example, one rule states that if super-concepts are the same, the actual concepts are similar to each other. NOM also exploits a set of instance-based techniques.

QOM. Quick Ontology Mapping (QOM) [226] is a successor of the NOM system [227]. The approach is based on the idea that the loss of quality in matching algorithms is marginal (to a standard baseline); however, improvement in efficiency can be tremendous. This fact allows QOM to produce mapping elements fast, even for large-size ontologies. QOM is grounded in matching rules of NOM. However, for the purpose of efficiency the use of some rules has been restricted. QOM avoids the complete pairwise comparison of trees in favor of an incomplete top-down strategy. Experimental study has shown that QOM is on par with other state-of-the-art algorithms for the quality of the proposed alignment, while outperforming them with respect to efficiency. Also, QOM shows better results than approaches within the same complexity class.

OLA. OWL Lite Aligner (OLA) [228] is designed with the idea of balancing the contribution of each component that composes an ontology (these include classes, properties, names, constraints, taxonomy, and even instances). As such it takes advantage of all the elementary matching techniques that have been considered in

the previous sections except the semantic ones. OLA is a family of distance-based algorithms which converts definitions of distances based on all the input structures into a set of equations. These distances are almost linearly aggregated (they are linearly aggregated modulo local matches of entities). The algorithm then looks for the matching between the ontologies that minimizes the overall distance between them. For that purpose it starts with base distance measures computed from labels and concrete datatypes. Then, it iterates a fix-point algorithm until no improvement is produced. From that solution, an alignment is generated which satisfies some additional criterion (on the alignment obtained and the distance between aligned entities). As a system, OLA considers the alignment as a solution to a clearly stated optimization problem.

Anchor-PROMPT. Anchor-PROMPT [234] (an extension of PROMPT) is an ontology-merging and alignment tool with a sophisticated prompt mechanism for possible matching terms. The anchor-PROMPT is a hybrid alignment algorithm which takes as input two ontologies (internally represented as graphs) and a set of anchor-pairs of related terms, which are identified with the help of string-based techniques (edit-distance test) or defined by a user, or another matcher computing linguistic similarity. Then the algorithm refines them by analyzing the paths of the input ontologies limited by the anchors in order to determine terms frequently appearing in similar positions on similar paths. Finally, based on the frequencies and user feedback, the algorithm determines matching candidates.

S-Match. S-Match [229] [230] [231] is a schema-based matching system. It takes two graph-like structures (e.g., XML schemas or ontologies) and returns semantic relations (e.g., equivalence, subsumption) between the nodes of the graphs that correspond semantically to each other. The relations are determined by analyzing the meaning (concepts, not labels) which is codified in the elements and the structures of schemas/ontologies. In particular, labels at nodes, written in natural language, are translated into propositional formulas which explicitly codify the label's intended meaning. This allows for a translation of the matching problem into a propositional unsatisfiability problem, which can then be efficiently resolved using (sound and complete) state-of-the-art propositional satisfiability deciders. S-Match was designed and developed as a platform for semantic matching, namely, as a highly modular system with a core of semantic relationship computations, where single components can be plugged, unplugged or suitably customized. It is a

hybrid system with a composition at the element level. At present, S-Match librar-
ies contains thirteen element-level matchers and three structure-level matchers.

Table 7.3. Summary of schema-matching approaches

	Element Level Matching		Structure Level Matching	
	Syntactic	**External**	**Syntactic**	**Semantic**
SF	string-based, datatypes, key properties		iterative fix-point computation	
Artemis	domain compatibility; language based	common thesaurus, broader term, related term	matching of neighbors via clustering	
Cupid	string-based, language-based, datatypes, key properties	auxiliary thesauri, synonyms, hypernyms, abbreviations	tree matching weighted by leaves	
COMA	string-based, language-based, datatypes	auxiliary thesauri, synonyms, hypernyms, abbreviations alignment reuse	DAG matching with bias toward children of leaf nodes; paths	
NOM/ QOM	string-based, domains and ranges	application specific vocabulary	neighbor matching, taxonomic structure	
Anchor-PROMPT	string-based, domain and ranges		bounded path matching	
OLA	string-based, language based, datatypes	WordNet	iterative fix-point computation, neighbor matching, taxonomic structure	
S-Match	string-based, language based	WordNet, sense-based, gloss-based		propositional SAT, DLs

7.3 Ontology Driven Information Integration

In order to achieve semantic interoperability in a heterogeneous information system, the meaning of the information that is interchanged has to be understood across the systems. Semantic conflicts occur whenever two contexts do not use the same interpretation of the information. The use of ontologies and metadata descriptions for the explication of implicit and hidden knowledge is a possible approach to overcome the problem of semantic heterogeneity. Uschold and Gruninger mention interoperability as a key application of ontologies, and many ontology-based approaches [239] to information integration in order to achieve interoperability have been developed.

In this section we discuss existing solutions for ontology-based information integration presented in [240]. Various approaches to intelligent information integration have been adopted in systems such as SIMS [241], TSIMMIS [242], OBSERVER [147], Carnot [21], InfoSleuth [243], KRAFT [244], PICSEL [245], DWQ [246], Ontobroker [247], SHOE [248], Crossvision Enterprise Information Integrator by Software AG [413] and others. Most of these systems use some notion of ontologies for integration across information resources. An evaluation of these approaches is presented based on the following criteria:

Use of Ontologies: The role and the architecture of ontologies heavily influence their representation formalism.

Ontology Representation: Depending on the use of the ontology, the representation capabilities differ from approach to approach.

Use of Mappings: In order to support the integration process the ontologies have to be linked to the underlying schemas used to store the data. If several ontologies are used in an integration system, inter-ontology mappings between classes in different ontoologies is also important.

Ontology Engineering: Before an integration of information sources can begin the appropriate ontologies have to be acquired or be selected for reuse. How does the integration approach support the acquisition or reuse of ontologies?

We begin with a discussion on various ontology-based architectures and the role of plau. This is followed by a discussion of the use of different representations, i.e., different ontology languages for information integration. Mappings used to connect ontologies to information sources, inter-ontology mappings, and associated methodologies for ontology engineering for information integration are also discussed.

7.3.1 The Role of Ontologies in Information Integration

Ontologies can be used in an integration task to explicitly describe the semantics of data an information stored in the underlying information sources. This can be achieved by identification of corresponding concepts from ontologies. The uses and roles played by ontologies in information integration is discussed next.

Explicit Semantic Descriptions

Different approaches for using ontologies for information integration can be characterized as: *single ontology approaches*, *multiple ontology approaches* and *hybrid ontology approaches*, and are illustrated in Figure 7.2. Some approaches provide a general framework where all three architectures can be implemented (e.g., DWQ [246]). A discussion of the three main architectures is as follows.
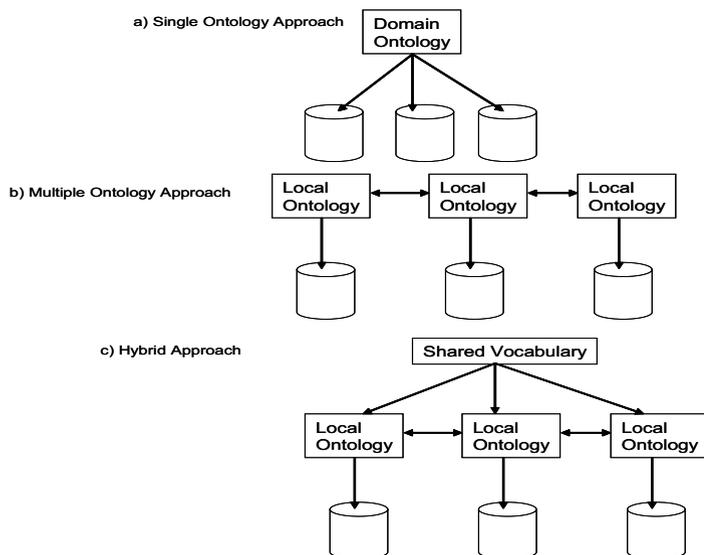

Fig. 7.2. Ontology-driven architectures for Information Integration

Single Ontology Approaches. Single ontology approaches use a domain ontology providing a shared vocabulary for the specification of the semantics. All information sources are related to one global ontology. We have adopted the single ontology approach in the solution design presented for the clinical use case and scenario. A prominent approach of this kind of ontology integration is SIMS [241]. The SIMS model of the application domain includes a hierarchical terminological knowledge base with nodes representing objects, actions, and states. An independent model of each information source is described for this system by relating the objects of each source to the global domain model. The relationships clarify the semantics of the source objects and help to find semantically corresponding objects.

Single ontology approaches can be applied to integration problems where all information sources to be integrated provide nearly the same view on a domain. But if one information source has a different view on a domain, e.g., by providing another level of granularity, finding the minimal ontology commitment [89] becomes a difficult task. For example, if two information sources provide product specifications but refer to absolute heterogeneous product catalogs which catego-

rize the products, the development of a global ontology which combines the different product catalogs becomes very difficult. Information sources with reference to similar product catalogs are much easier to integrate. Also, single ontology approaches are susceptible to changes in the information sources which can affect the conceptualization of the domain represented in the ontology. Depending on the nature of the changes in one information source it can imply changes in the global ontology and in the mappings to the other information sources. These disadvantages led to the development of multiple ontology approaches.

The domain ontology can also be a combination of several specialized ontologies. A reason for the combination of several ontologies can be the modularization of a potentially large monolithic ontology. The combination is supported by ontology representation formalisms, i.e., by importing other ontology modules (e.g., Ontolingua [89]).

Multiple Ontology Approaches. In multiple ontology approaches, each information source is described by its own domain- or application-specific ontology. For example, in OBSERVER [147], the semantics of an information source is described by a domain-specific ontology. In principle, the domain ontology can be a combination of several other ontologies but it cannot be assumed that the different domain ontologies share the same vocabulary. The Crossvision Information Integrator supports a multiple ontology approach. It relies on information models which are organized using ontologies, which are managed in a metadata repository (CentraSite). A semantic inference engine (Semantic Server) then allows the raw data to be aggregated dynamically, perfectly tailored for the individual business user's needs.

At a first glance, the advantage of multiple ontology approaches seems to be that no common and minimal ontology commitment around an ontology is needed. Each ontology could be developed without respect to other information sources or domain ontologies — no common ontology with the agreement of all information sources/ontologies are needed. This ontology architecture can simplify the change, i.e., modifications in one information source/ontology or the adding and removing of information sources/ontologies. But in reality the lack of a common vocabulary makes it extremely difficult to compare different source ontologies. To overcome this problem, an additional representation formalism defining the inter-ontology mapping is required. The inter-ontology mapping identifies semantically corresponding terms of different ontologies, terms which are semantically equal or similar. But the mapping also has to consider different views on a domain, i.e., different granularities of the ontology concepts. Issues of semantic heterogeneity may also occur in defining inter-ontology mappings.

Hybrid Ontology Approaches. To overcome the drawbacks of the single or multiple ontology approaches, hybrid approaches were developed. Similar to multiple ontology approaches the semantics of each source is described by its appropriate domain ontology. But in order to make the source ontologies comparable to each other they are built upon one global shared vocabulary [249] [250]. The shared vocabulary contains basic terms (the primitives) of a domain. In order to

build complex terms of ontologies the primitives are combined by some operators. Because each term of an ontology is based on the primitives, the terms can be easily mapped to each other, than in multiple ontology approaches. Sometimes the shared vocabulary is also an ontology.

In the COIN system [249], the local description of a piece of information, the so-called context, is simply an attribute value vector. The terms for the context stem from the common shared vocabulary and the data itself. In the MECOTA system [250] each piece of source information is annotated by a label which indicates the semantics of the information. The label combines primitive terms from the shared vocabulary. The combination operators are similar to the operators known from the description logics, but are extended for the special requirements resulting from integration of sources, e.g., by an operator for aggregation. In the BUSTER system [251], the shared vocabulary is a (general) ontology, which covers all possible refinements. For example, the general ontology defines the attribute value ranges of its concepts. A domain ontology is one (partial) refinement of the general ontology, e.g., restricting the value range of some attributes. Since domain ontologies only use the general ontology, they remain comparable.

The advantage of a hybrid approach is that new sources can easily be added without the need of modification in the mappings or in the shared vocabulary. It also supports the acquisition and evolution of ontologies. The use of a shared vocabulary makes the source ontologies comparable and avoids the disadvantages of multiple ontology approaches. The drawback of hybrid approaches, however, is that existing ontologies cannot be reused easily, but have to be redeveloped from scratch, because all domain ontologies have to refer to the shared vocabulary.

Ontologies as a Query Model

Integrated information sources normally provide an integrated view. Some integration approaches use the ontology as the query schema, e.g., the SIMS system [241]. The user formulates a query in terms of the ontology. The system reformulates the query into subqueries for each appropriate source, collects and combines the query results, and returns the results. Using an ontology as a query model has the advantage that the structure of the query model should be more intuitive for the user because it corresponds more to the user's appreciation of the domain. However, the user has to know the structure and the contents of the ontology.

Ontologies as Verification Mechanism

During the integration process several mappings must be specified from a domain ontology to the local source schema. The correctness of such mappings can be considerably improved if these can be verified automatically. A subquery is correct with respect to a query if the local subquery provides a part of the queried answers, i.e., the subqueries must be contained in the global query (query containment) [246][245]. Since an ontology contains a (complete) specification of the conceptu-

alization, the mappings can be validated with respect to these ontologies. Query containment means that the ontology concepts corresponding to the local subqueries are contained in the ontology concepts related to the query.

In the DWQ system [246], each source is assumed to be a collection of relational tables. Each table is described in terms of its ontology with the help of conjunctive queries. A query and the decomposed subqueries can be unfolded to their ontology concepts. The subqueries are correct, i.e., are contained in the query, if their ontology concepts are subsumed by the ontology concepts. The PICSEL project [245] can also verify the mapping, but in contrast to DWQ it can also generate mapping hypotheses automatically which are validated with respect to a global ontology.

The quality of the verification strongly depends on the completeness of an ontology. If the ontology is incomplete, the verification result can erroneously imply a correct query subsumption. Since in general the completeness can not be measured, it is impossible to make any statements about the quality of the verification.

7.3.2 Ontology Representations Used in Information Integration

Various approaches to intelligent information integration based on ontologies have predominantly used variants of description logics in order to represent ontologies. The CLASSIC system [165] has been used in the OBSERVER system [148] and by database researchers investigating semantic heterogeneities and interoperability [252]. The SIMS system makes use of the LOOM description logic [253]. Other terminological languages used are GRAIL [254], used in the TAMBIS system [46], and OIL [256], which is used for terminology integration in the BUSTER system [255].

Besides the purely terminological languages mentioned above there are also approaches using extensions of description logics which include rule bases. Some examples are the use of CARIN [167], a description logic extended with function-free horn rules in the PICSEL system [245]. The DWQ project [246] uses AL-log [257], which combines simple description logics with Datalog and the logic DLR, a description logic with n-ary relations. The integration of description logics with rule-based reasoning makes it necessary to restrict the expressive power of the terminological part of the language in order to maintain decidability.

The second main group of languages used in ontology-based information integration systems are classical frame-based representation languages. Examples for such systems are COIN [249], KRAFT [244] and InfoSleuth [243]. There are also approaches that directly use F-Logic [82] with a self-defined syntax (Ontobroker [247] and COIN [249]).

7.3.3 The Role of Mapping in Information Integration

The task of integrating heterogeneous information sources provides a use case for ontologies. Ontologies may be viewed as the glue that puts together information of various kinds. Mappings refer to the connection of an ontology to other parts of the system. Mapping are a critical requirement for information integration for (a) connecting ontologies with the information source they describe; and (b) connecting different ontologies used in a system. In Section 7.2, we discussed a representative set of techniques to identify and discover mappings between two schema or ontology like artifacts. In this section, we discuss how these mappings, once generated can be used in the context of Information Integration.

Mapping Ontologies to Information Resources

Different approaches used to establish a connection between ontologies and information sources are as follows.

Structure Resemblance. A straightforward approach to connecting an ontology with the database schema is to simply produce a one-to-one copy of the structure of the database and encode it in a language that makes automated reasoning possible. The integration is then performed on the copy of the model and can easily be tracked back to the original data. This approach is implemented in the SIMS mediator [258] and also by the TSIMMIS system [242].

Definition of Terms. In order to make the semantics of terms in a database schema clear it is not sufficient to produce a copy of the schema. There are approaches such as those used in the BUSTER system [255] that use the ontology to further define terms from the database or the database schema. These definitions can consist of a set of rules defining the term and are, in most cases, described by concept definitions.

Structure Enrichment. This is the most common approach for relating ontologies to information sources, and combines the two previous approaches. A logical model is built that resembles the structure of the information source and contains additional definitions of concepts. A detailed discussion of structure is presented in [252], which is used in OBSERVER [147], KRAFT [244], PICSEL [245] and DWQ [246]. While OBSERVER uses description logics for both structure resemblance and additional definitions, PICSEL and DWQ define the structure of the information by (typed) horn rules. Additional definitions of concepts mentioned in these rules are given by a description logic model. KRAFT does not commit to a specific definition scheme.

Meta-annotation. An interesting approach is the use of meta-annotations that add semantic information to an information source. This approach is particularly relevant in the context of the integrating information on the Web, where annotation may be viewed as a natural way of adding semantics. Ontology-based integration approaches developed for the Web context are the Ontobroker [247] and SHOE [248] systems.

Inter-ontology Mapping

Some information integration systems such as [148] [244] use more than one ontology to describe the information. The problem of mapping different ontologies is a well-known problem in knowledge engineering. We now discuss approaches that are used in the context of information integration systems.

Defined Mappings. In the KRAFT System [244], translations between different ontologies are done by special mediator agents which can be customized to translate between different ontologies and even different languages. Different kinds of mappings are distinguished in this approach starting from simple one-to-one mappings between classes and values to mappings between compound expressions. This approach allows great flexibility, but it fails to ensure a preservation of semantics: the user is free to define arbitrary mappings even if they do not make sense or produce conflicts.

Lexical Relations. An attempt to provide at least intuitive semantics for mappings between concepts in different ontologies is made in the OBSERVER system [148]. The approaches extend a common description logic model by quantified inter-ontology relationships borrowed from linguistics. The relationships used are synonym, hypernym, hyponym, overlap, covering and disjoint. While these relations are similar to constructs used in description logics they do not have a formal semantics. Consequently, the query translation algorithm is probabilistic in nature.

Top-Level Grounding. In order to avoid a loss of semantics, one has to stay inside the formal representation language when defining mappings between different ontologies (e.g., DWQ [Calvanese et al., 2001]). A straightforward way to achieve this is to relate all ontologies used to a single top-level ontology. This can be done by inheriting concepts from a common top-level ontology. This approach can be used to resolve conflicts and ambiguities. While this approach enables establishment of connections between concepts from different ontologies in terms of common superclasses, it does not establish a direct correspondence. This might lead to problems when exact matches are required.

Semantic Correspondences. An approach that tries to overcome the ambiguity that arises the previous approach, is to identify well-founded semantic correspondences between concepts from different ontologies. In order to avoid arbitrary mappings between concepts, these approaches have to rely on a common vocabulary for defining concepts across different ontologies. One approach uses semantic labels in order to compute correspondences between database fields. Another approach is to represent concepts from different ontologies in a description logic model of terms and use subsumption reasoning to establish relations between different terminologies. Approaches using formal concept analysis also fall into this category, because they define concepts on the basis of a common vocabulary to compute a common concept lattice.

7.3.4 The Role of Ontology Engineering in Information Integration

Since ontologies play a crucial role semantic information integration, it is crucial to support the ontology engineering process, especially that part which is likely to have an impact on the information integration process.

Ontology Development Methodologies

Example information integration systems and their approaches for developing ontologies are discussed as follows.

InfoSleuth. Ontologies in InfoSleuth are defined primarily manually using Entity-Relationship (E-R) models. Approaches for semi-automatic construction of ontologies from textual databases have been proposed in [259]. The methodology is as follows: first, human experts provide a small number of seed words to represent high-level concepts. The system then processes the incoming documents, extracting phrases that involve seed words, generates corresponding concept terms, and classifies them into the ontology. During this process the system also collects seed word candidates for the next round of processing. This iteration can be completed for a predefined number of rounds. A human expert verifies the classification after each round. As more documents arrive, the ontology expands and the expert is confronted with the new concepts. This is a significant feature of this system, called the "discover and alert" feature.

KRAFT. Ontologies in KRAFT are built based on two methods: manual construction of shared ontologies and extraction of domain or information source ontologies. KRAFT offers two methods for building ontologies:

- The steps of the development of shared ontologies are (a) ontology scoping, (b) domain analysis, (c) ontology formalization and (d) top-level ontology. The minimal scope is a set of terms that is necessary to support the communication within the KRAFT network. The domain analysis is based on the idea that changes within ontologies are inevitable and the means to handle changes should be provided. The authors pursue a domain-led strategy, where the shared ontology fully characterizes the area of knowledge in which the problem is situated. Within the ontology formalization phase the fully characterized knowledge is defined formally in classes, relations and functions. The top-level ontology is needed to introduce predefined terms/primitives.
- A bottom-up approach to extract an ontology from existing shared ontologies was introduced in [260]. The first step is a syntactic translation from the KRAFT exportable view (in a native language) of the resource into the KRAFT schema. The second step is the ontological upgrade, a semi-automatic translation plus knowledge-based enhancement, where the local ontology adds knowledge and further relationships between the entities in the translated schema.

Ontobroker. There are three classes of Web information sources [261]: (a) Multiple-instance sources with the same structure but different contents, (b) single-

instance sources with large amount of data in a structured format, and (c) loosely structured pages with little or no structure. Ontobroker uses two ways of formalizing knowledge. First, sources from (a) and (b) allow it to implement wrappers that automatically extract factual knowledge from these sources. Second, sources with little or no knowledge have to be formalized manually.

SIMS. An independent model of each information source is described in the SIMS system, along with a domain model that must be defined to describe objects and actions. The SIMS model of the application domain includes a hierarchical terminological knowledge base with nodes representing objects, actions, and states. In addition, it includes indications of all relationships between the nodes. Scalability and maintenance problems on addition of a new information source or change in domain knowledge are addressed. As every information source is independent and modeled separately, the addition of a new source is relatively straightforward. A graphical LOOM knowledge base builder (LOOM-KB) is used to support this process. The domain model is enlarged to accommodate new information sources or new knowledge.

Tools for the Annotation Process

Some of the systems discussed in this chapter provide support with the annotation process of information sources, leading to a semantic enrichment of the information. Some tools used in the process are OntoStudio (previously known as Onto-edit, discussed in Chapter 6.3), the SHOE Knowledge Annotator and the I-COM tool used in the DWQ project. With the help of the SHOE Knowledge Annotator tool, the user can describe the contents of a Web page [262]. The Knowledge Annotator has an interface which displays instances, ontologies, and claims (documents collected). The tool also provides integrity checks. With a second tool called Expose the annotated Web pages are parsed and the contents stored in a repository. The I-COM tool [111] was developed within the DWQ project. This tool uses an extended entity-relationship (EER) conceptual data model and enriches it with aggregations and inter-schema constraints.

Ontology Evolution

Support for ontology evolution is a critical piece of functionality in the context of an information integration system. An integration system and the ontologies must support adding and/or removing sources and must be robust to changes in the information source. The SHOE system is one system that takes these issues into account.

Once the SHOE-annotated Web pages are uploaded on the Web, the Expose tool has the task to update the repositories with the knowledge from these pages. This includes a list of pages to be visited and an identification of all hypertext links, category instances, and relation arguments within the page. The tool then stores the new information in the PARKA knowledge base. The problems associated with

managing dynamic ontologies through the Web have been presented in [248]. By adding revision marks to the ontology, changes and revision become possible. The authors illustrated that revisions which add categories and relations will have no effect, and that revisions which modify rules may change the answers to queries. When categories and relations are removed, answers to queries may be eliminated.

7.4 Summary

In this chapter we presented applications of metadata and ontologies, such as semantic annotations, mappings and information integration. These applications are enabled by semantic descriptions of data and resources on the web-based and other repositories; and themselves enable new functionality on the web and on internal organizations' intranets. We presented tools and techniques for annotation of Web resources with semantic metadata annotations. Two types of metadata data annotations are considered: (a) structured and semi-structured metadata annotations of unstructured Web content; and (b) structured metadata annotations of structured Web content. It was noted that the latter corresponds to mapping the schemas underlying the structured content to domain-specific ontologies, and a discussion and taxonomy of schema-matching techniques was also presented. Finally, we presented various approaches adopted for ontology driven information integration, including a discussion on various types of architectures, the role played by ontologies in the creation of mappings, specifying queries and as a verification mechanisms.