6 Ontology Authoring and Management

Ontologies are a critical component of the Semantic Web architecture. In this chapter, we present a discussion on various aspects of ontology authoring and management. We discuss a collection of ontology building tools and present an evaluation across various dimensions. A brief discussion on techniques for boostrapping of ontologies is presented along with a discussion on techniques of integration, merging and versioning of ontologies.

6.1 Ontology Building Tools

We begin with a survey of ontology building and editing tools that are in use today [99] [100]. The tools may be useful for building ontology schemas (terminological components) alone or together with instance data. Ontology browsers without an editing focus and other types of ontology building tools are not included. Concise descriptions of each software tool are presented and compared according to different criteria which are presented below.

Software architecture and tool evolution, which includes information about the tool architecture (standalone, client/server, n-tier application), how the tool can be extended with other functionalities/modules, how ontologies are stored (databases, text files, etc.) and if there is any backup management system.

Interoperability with other ontology development tools and languages, which includes information about the interoperability capabilities of the tool. We will review the tool's interoperability with other ontology tools (for merge, annotation, storage, inferencing, etc.), as well as translations to and from ontology languages.

Knowledge representation. We will present the KR paradigm underlying the knowledge model of the tool. It is very relevant in order for us to know what and how knowledge can be modeled in the tool. We will also analyze if the tool provides any language for building axioms.

Inference services attached to the tool. We will analyze whether the tool has a built-in inference engine or it can use other external inference engines. We will also analyze if the tool performs constraint/consistency checking, if it can automatically classify concepts in a concept taxonomy and if it is able to manage exceptions in taxonomies.

Usability. We will analyze the existence of graphical editors for the creation of concept taxonomies and relations, the ability to prune these graphs and the possi-

bility to perform zooms of parts of it. We will also analyze if the tool allows some kind of collaborative working and if it provides libraries of ontologies.

6.1.1 Ontology Editors: Brief Descriptions

A brief description of some of the prominent ontology editors in use today is presented below.

Apollo: Apollo [101] is a user-friendly ontology development application, motivated by requirements of industrial users who wished to use knowledge-modeling techniques, but require a syntax and an environment that is easy to use. The application is implemented in Java and supports all the basic primitives of knowledge modeling: ontologies, classes, instances, functions and relations. Full consistency checking is done while editing, for example, detecting the use of undefined classes. Apollo has its own internal language for storing the ontologies, but can also export the ontology into different representation languages, as required by the user.

LinKFactory: Link Factory [102] LinKFactory® is a formal ontology management system developed by Language & Computing NV, designed to build and manage very large and complex language-independent formal ontologies. The LinKFactory system consists of 2 major java-based components: the LinKFactory® Server, and the LinKFactory Workbench (client side component). The LinkFactory Workbench allows the user to browse and model several ontologies and align them. From the knowledge representation and underlying reasoning point of view, LinkFactory has the following characteristics and possibilities: fixed built-in ISA (formal subsumption), DISJOINT, and SAME-AS relationships, definable relationship hierarchy (multiple hierarchies), specification of necessary and sufficient conditions for individual concept definitions, several constraint-checking methods, autoclassification of new concepts on the basis of natural language terms as well as formal definitions, mechanisms to map and/or merge various ontologies; and automatic text analysis to assign links to the ontology.

OntoStudio: OntoStudio [103], the successor of OntoEdit, is a engineering environment for ontology creation and maintenance. OntoStudio is built on top of a powerful internal ontology model. This paradigm supports representation-language-neutral modeling as much as possible for concepts, relations and axioms. Several graphical views on structures in the ontology support representation of different phases of the ontology engineering cycle. The tool allows the user to edit a hierarchy of concepts or classes. A concept may have several names, which essentially is a way to define synonyms for that concept. The tool for reorganization of concepts within the hierarchy is based on a "copy-and-paste" like functionality. The tool is based on a flexible plug-in framework, allowing easy extension and customization. Some available plug-ins are: (a) inferencing for consistency checking, classification and execution of rules; (b) collaborative engineering of ontologies; and (c) an ontology server for administration, collaborative sharing of and persistent storage for ontologies.

Ontolingua Server: The Ontolingua Server is a set of tools and services that support the building of shared ontologies between distributed groups, and that have been developed by the Knowledge Systems Laboratory (KSL) at Stanford University. The ontology server architecture provides access to a library of ontologies, translators to languages (Prolog, CORBA IDL, CLIPS, Loom, etc.) and an editor to create and browse ontologies. Remote editors can browse and edit ontologies, and remote or local applications can access any of the ontologies in the ontology library using the OKBC (Open Knowledge-Based Connectivity) protocol.

Ontosaurus: Ontosaurus [105], developed by the Information Sciences Institute (ISI) at the University of South California, consists of two modules: an ontology server, which uses Loom as its knowledge representation language, and an ontology browser server that dynamically creates HTML pages (including image and textual documentation) that displays the ontology hierarchy. The ontology can be edited by HTML forms, and translators exist for translation from LOOM to Ontolingua, KIF, KRSS and C++.

Protege: Protégé [106] is an open source ontology editor that has seen wide usage from modeling cancer protocol guidelines to nuclear power stations. Protégé provides a graphical and interactive ontology design and knowledge base development environment. Tree controls allow quick and simple navigation through a class hierarchy. Protégé uses forms as the interface for filling in slot values. The knowledge model of Protégé is OKBC compatible, and includes support for classes and the class hierarchy with multiple inheritance; template and own slots; specification of pre defined and arbitrary facets for slots, which include allowed values, cardinality restrictions, default values, and inverse slots; and metaclasses and metaclass hierarchy. The Protege architecture supports a database backend and caching mechanism. Its component-based architecture enables system builders to add new functionality by creating appropriate plug-ins, which fall into one of the three categories: (1) backends for import and export of knowledge bases in various formats; (2) slot widgets for display and edit of slot values in domain and task specific ways; and (3) tab plug-ins which are typically applications tightly linked with Protege knowledge bases. Current back-end plug-ins include export and import from-RDF Schema, XML Schema and OWL files. Currently, tabs that enable advanced visualization, ontology merging, version management and inferences are available.

WebODE: WebODE [107] [108] is engineering workbench that provides services for the ontology development process. Ontologies are represented using a very expressive knowledge model, based on the reference set of intermediate representations of the METHONTOLOGY methodology [109], which includes ontology components such as concepts (with instance and class attributes), partitions, ad hoc binary relations, predefined relations (taxonomic and part-of), instances, axioms, rules, constants and bibliographic references. It also allows the import of terms from other ontologies. Ontologies in WebODE are stored in a relational database underlying a well-defined service-oriented API for ontology access that makes easy integration with other systems. Ontologies built with WebODE can be easily integrated with other systems by using its automatic export/import services

from and into XML, and translation services into and from various ontology specification languages (currently, RDF(S), OWL, CARIN and FLogic) and systems such as Java and Jess. Authoring is aided both by form-based and graphical user interfaces, a user-defined views manager, a consistency checker, an inference engine, an axiom builder and the documentation service. Two interesting and novel features of WebODE are: instance set for instantiating the same conceptual model for different scenarios, and conceptual views from the same conceptual model. The graphical user interface allows browsing all the relationships defined on the ontology as well as pruning these views with respect to selected types of relationships. WebODE also supports collaborative authoring of ontologies. Constraint-checking capabilities are also provided for type constraints, numerical values constraints, cardinality constraints and taxonomic consistency verification.

WebOnto: WebOnto [110] is a tool developed by the Knowledge Media Institute (KMi) of the Open University (England). It supports the collaborative browsing, creation and editing of ontologies, which are represented in the knowledge-modeling language OCML. Its main features are: management of ontologies using a graphical interface; the automatic generation of instance editing forms from class definitions; support for Problem Solving Methods (PSMs) and task modeling; inspection of elements incorporating inheritance of properties and consistency checking; a full tell and ask interface, and support for collaborative work; by means of broadcast/receive; and making annotations.

ICOM: ICOM [111] supports the conceptual design phase of an information system. An Extended Entity-Relationship (EER) conceptual data model, enriched with multidimensional aggregations and inter-schema constraints, is used. ICOM is fully integrated with a very powerful description logic reasoning server which acts as a background inference engine. The ICOM modeling language can express: (a) the standard E-R data model, enriched with IsA links (i.e., inclusion dependencies), disjoint and covering constraints, full-cardinality constraints, and definitions attached to entities and relations by means of view expressions over other entities and relationships in the schema; (b) aggregated entities together with their multiply hierarchically organized dimensions; and (c) a rich class of (inter-schema) integrity constraints, as inclusion and equivalence dependencies between view expressions involving entities and relationships possibly belonging to different schemas. ICOM reasons with (multiple) diagrams by encoding them in a single description logic knowledge base, and shows the result of any deductions such as inferred links, new stricter constraints, and inconsistent entities or relationships. The DLR description logics is used to encode the schemas and to express the views and the constraints. The Java-based tool allows for the creation, the editing, the managing, and the storing of several interconnected conceptual schemas, with a user-friendly graphical interface (including an auto-layout facility).

IODE: OntologyWorks IODE [112] is a data and information modeling tool for creating high-definition ontologies, specifically designed for supporting ontology development for database and application development. IODE incorporates a library of vetted and comprehensive domain-independent content that knits

together and guides the development of ontologies, ensuring interoperability across domains and across time. It uses a powerful logic (SCL) for representation of ontologies and verifies the consistency of these ontologies, both internally and with respect to domain-independent content. IODE supports existing W3C standards such as RDF and OWL, and supports management of ontology versions in a transactional environment.

Visual Ontology Modeler: The Visual Ontology Modeler [113] by Sandpiper Software is a visual application for building component-based ontologies. It is a UML-based modeling tool that enables ontology development and management for use in collaborative applications and interoperability solutions. Some key features are: (a) A multi-user, network-based environment for ontology development in a rich, graphical notation; (b) Automated import/export facilities in XML schema, RDF, OWL, DAML, and MOF formats; (c) A feature-rich set of ontology authoring wizards that create and maintain the required UML model elements for the user, saving time and substantially reducing construction errors and inconsistencies. The Visual Ontology Modeler implements Sandpiper's UML Profile for Knowledge Representation, which extends UML to enable modeling of frame-based knowledge representation concepts such as class, relation, function and individual frames, as well as the slots and facets that constrain those frames. It also includes a library of ontologies, including the IEEE Standard Upper Ontology (SUO), concepts relevant to XML schema, RDF, and DAML generation, and other basic concepts to develop rich ontologies. The framework supports analysis, alignment, development, merging, and evolution, with consistency checking and validation for OWL-DL, first order, and production rules-related applications.

Semtalk: Semtalk [114] is a Microsoft Visio based graphical modeling tool, which is used for business process modeling, product configuration and visual glossaries. Since it is based on an open extensible meta-model, new modeling tools can be created with reasonable effort. Most of these solutions make use of SemTalk's ability to represent ontologies or at least taxonomies in a visual way using Microsoft Visio. The native modeling language supported by the SemTalk consistency engine is a mixture of RDF(S) and OWL. It supports multiple inheritance, instances, and object and datatype properties. A UML-based graphical representation is adopted for the graphical representation of ontologies. Semtalk provides export and import interfaces to RDF(S), OWL, DAML and F-Logic.

COBra: COBra [115] is an ontology browser and editor for GO and OBO ontologies. It has been specifically designed to be usable by biologists to create links between ontologies, and supports: (a) drag-and-drop editing of GO ontologies; (b) mapping between two ontologies; and (c) translation to OWL and other Semantic Web languages. COBra supports manual creation of links between terms in two ontologies, e.g.,links or mappings between tissues in an anatomy and the cell types of the tissues can be recorded and stored. COBra supports import/export of ontologies from and into GO and GO XML/RDF/RDF(S), DAG Edit and OWL.

Generic Knowledge Base (GKB) Editor: The GKB Editor [116] is a tool for graphically browsing and editing knowledge bases across multiple frame represen-

tation systems (FRSs) in a uniform manner. It offers an intuitive user interface in which objects and data items are represented as nodes in a graph, with the relationships between them forming the edges. A sophisticated incremental browsing facility allows the user to selectively display only that region of a KB that is currently of interest, even as that region changes. The GKB Editor consists of three main modules: a graphical interactive display based on Grasper-CL, and a library of generic knowledge-base functions, and corresponding libraries of frame-representation-specific methods, based on Open Knowledge Base Connectivity (OKBC).

SWOOP: Most existing ontology development toolkits provide an integrated environment to build and edit ontologies, check for errors and inconsistencies (using a reasoner), browse multiple ontologies, and share and reuse existing data by establishing mappings among different ontological entities. However, their UI design (look and feel) and usage style are inspired by traditional KR-based paradigms, whose constrained and methodical framework have steep learning curves, making them cumbersome to use for the average Web user. SWOOP [117] is a hypermedia-inspired ontology editor that employs a Web browser metaphor for its design and usage. Such a tool would be more effective (in terms of acceptance and use) for the average web user by presenting a simpler, consistent and familiar framework for dealing with entities on the Semantic Web. Some features of SWOOP are: (a) Web-browser-like look and feel including URI-based access and hyperlink-based navigation; (b) inline editing with HTML renderer; (c) browsing, comparison and mapping of multiple ontologies; (d) ontology partitioning and explanation; (e) collaborative annotation support; and (e) sound and complete conjunct Abox queries.

WSMT: The Web Services Modeling Toolkit [377] is an open source graphical development environment for all elements of the Web Service Modeling Ontology (WSMO). It is built as a set of plug-ins for the Eclipse development environment and allows ontology engineers to graphically build their ontologies. Particular focus is placed on visualization of large ontologies with zoom-in and zoom-out capabilities. Verification and consistency checking are provided through a plugged-in WSML reasoner. Another plug-in allows the creation of mappings between ontologies that can be used in a WSMO execution environment.

WSMO Studio: WSMO Studio [378] also provides an open source Eclipse-based development environment for building WSMO ontologies. Similarly to WSMT, reasoning support can be plugged in for different WSML language variants. It also directly supports WSMO annotations of existing WSDL Web Service descriptions via the W3C SAWSDL. Another difference with WSMT is that WSMO Studio does not focus on graphical visualization of ontologies.

TopBraid Composer: TopBraid Composer is an enterprise-class modeling environment for developing Semantic Web ontologies and building semantic applications. Fully compliant with W3C standards, Composer offers comprehensive support for developing, managing and testing configurations of knowledge models and their instance knowledge bases. Composer incorporates a flexible and extensi-

ble framework with a published API for developing semantic client/server or browser-based solutions, that can integrate disparate applications and data sources. Implemented as an Eclipse plug-in, Composer is used to develop ontology models, configure data source integration as well as to customize dynamic forms and reports. Other than W3C standards, there is support for importing UML models, XML Schemas and relational databases. It supports integration with leading RDF data stores such as Jena, Pellet and Racer.

Neon Toolkit: The NeOn toolkit [406], based on the OntoStudion Editor core, is an extensible Ontology Engineering Environment. It contains plugins for ontology management and visualization. The core features of the Neon toolkit include support for basic schema editing operations, visualization and browsing of ontologies, the ability to import and export ontologies in various representation languages such as F-Logic, subsets of RDF(S) and OWL. It is designed around an open and modular architecture, which includes infrastructure services such as registry and repository, and supports distributed components for ontology management, reasoning and collaboration in networked ontologies. Building on the Eclipse platform, the Toolkit provides an open framework for plug-in developers. A number of commercial plugins are available that extent the toolkit by various functionalities including support for rules, development and interpretation of mappings, ability to access databases and import database schemas and specify queries in a Query-Editor.

6.1.2 Ontology Editors: A Comparative Evaluation

We present a comparative evaluation based on the dimensions identified earlier.

Table 6.1. Ontology editing tools: Architecture

Tool	SW Architecture	Extensibility	Ontology Storage	Backup Management
Apollo	Standalone	Plug-ins	Files	No
LinKFactory	3-tier	Plug-ins	DBMS	No
OntoStudio	Client Server	Plug-ins	DBMS	No
Ontolingua Server	Client Server	No	Files	No
Ontosaurus	Client Server	No	Files	No
Protege	Standalone	Plug-ins	DBMS	No
WebODE	3-tier	Plug-ins	DBMS	Yes
WebOnto	Client Server	No	Files	Yes
ICOM	Client Server	No	XML Files	
IODE	Standalone	No	Deductive DBMS	Yes

Table 6.1. Ontology editing tools: Architecture

Tool	SW Architecture	Extensibility	Ontology Storage	Backup Management
Visual Ontology Modeler	Plug-in to Rational Rose	Yes	As UML class diagram	Yes
Semtalk	Plug-in for Microsoft Visio	No	Visio files	No
COBra	Standalone	No	Flat files (multiple formats)	No
GKB	Standalone and client server	No	Yes	No
SWOOP	Web-based client server	Yes via plug-ins	As HTML models	No
WSMT	Standalone Eclipse plug-in	Plug-ins	Flat file	No
WSMO Studio	Standalone Eclipse plug-in	Plug-ins	Flat file	No
Topbraid Composer	Standalone Eclipse plug-in	Plug-ins	DBMS	Yes
Neon Toolkit	Standalone Eclipse plug-in	Plug-ins	DBMS	Yes

Table 6.1 presents a comparison of various ontology editors and tools based on its software architecture (standalone, client/server, n-tier application), extensibility, storage of the ontologies (databases, ASCII files, etc.) and backup management. From this perspective, most of the tools are moving toward Java platforms, and most of them are moving to extensible architectures as well. Storage in databases is and backup management are weak points of ontology tools.

Interoperability (Table 6.2) with other ontology development tools, merging tools, information systems and databases, as well as translations to and from some ontology languages, are important for integration of ontologies into applications. Most of the new tools export and import to adhoc XML and other markup languages.

Table 6.2. Ontology editing tools: Knowledge representation and methodological support

Tool	KR Knowledge Model	Axiom Language	Methodological Support
Apollo	Frames (OKBC)	Unrestricted	No
LinKFactory	Frames + First Order Logic	Restricted First Order Logic	Yes
OntoStudio	Frames + First Order Logic	FLogic	OntoKnowledge
Ontolingua Server	Frames + First Order Logic	KIF	No
Ontosaurus	Description Logics	LOOM	No
Protege	Frames + First Order Logic + Metaclasses	PAL	No
WebODE	Frames + First Order Logic	WAB	Methontology
WebOnto	Frames _ First Order Logic	OCML	No
ICOM	Description Logics with extension	DLR	No
IODE	Common Logic, extended with temporal reasoning and quantification over predicates	FOL	Yes
Visual Ontology Modeler	Description Logics	DL	Own - collaborative ontology development
Semtalk	OWL	OWL Full is possible	No
COBra	RDF and OWL	Not used	No
GKB	Multiple Frame Representation Systems	LOOM and others	No
SWOOP	OWL	OWL-DL	No
WSMT	WSML	WSML	No
WSMO Studio	WSML	WSML	No
Topbraid Composer	RDF, OWL and SWRL	OWL-DL	No
Neon Toolkit	F-Logic, RDF and OWL	F-Logic, OWL-DL	Yes, Neon Ontology Dev Process and Lifecycle

From the knowledge representation point of view (Table 6.2), there are two families of tools: description-logic-based tools, and other tools, which allow representation of knowledge following a hybrid approach based on frames and first order logic. Additionally, Protégé provides flexible modeling components like metaclasses. Some ontology building methodologies that are supported are: the On-toKnowledge methodology, GALEN methodology and Methontology. None of the tools provide project management facilities, and provide only a little support for ontology maintenance and evaluation.

Table 6.3. Ontology editing tools: Inference services

Tool	Inbuilt Inference Engine	External Inference Engine	Constraint, Consistency Checking	Automatic Classification	Exception Handling
Apollo	No	No	Yes	No	No
LinKFac-tory	Yes	Yes	Yes	Yes	No
OntoStudio	Yes (Ontobro-ker)	No	Yes	No	No
Ontolingua Server	No	ATP	No	No	No
Ontosaurus	Yes	Yes	Yes	Yes	No
Protege	Yes (PAL)	Jess, FLogic, Pellet	Yes	No	No
WebODE	Yes (Pro-log)	Jess	Yes	No	No
WebOnto	Yes	No	Yes	No	No
ICOM	Not in GUI	Connect to ICOM server	Yes	No	No
IODE	Yes	No	Yes	No	Yes
Visual Ontology Modeler	No	DL rea-soner, rules engines	Yes	No	No
Semtalk	No	No	No	No	No
COBra	No	No	No	No	No
GKB	No	Yes	Yes	No	No

Table 6.3. Ontology editing tools: Inference services

Tool	Inbuilt Inference Engine	External Inference Engine	Constraint, Consistency Checking	Automatic Classification	Exception Handling
SWOOP	No	Yes (Pellet or other engine)	Only with reasoner plug-in	No	No
WSMT	No	Yes, via plug-in	Yes	No	No
WSMO Studio	No	Yes, via plug-in	Yes	No	No
Topbraid Composer	No	Yes, OWLIM, Pellet, Jena, Oracle Rules	Yes	Yes	Yes
Neon Toolkit	Yes, Ontobroker	Yes, KAON-2 Engine	Yes	No	Yes

Before selecting a tool, it is also important to know which inference services are attached to it (Table 6.3). This includes built-in and other inference engines, constraint and consistency-checking mechanisms, automatic classifications and exception handling, among others. LinkFactory has its own inference engine, OntoStudio uses OntoBroker, Ontolingua uses ATP, Ontosaurus uses the Loom classifier, Protégé uses PAL and can also be linked to DL reasoners, WebODE uses Ciao Prolog and WebOnto uses the OCML inference engine. Besides, WebODE and Ontosaurus provide evaluation facilities. LinkFactory performs automatic classification. Finally, none of the tools provide exception-handling mechanisms.

Table 6.4. Ontology editing tools: Usability

Tool	Graphical Taxonomy	Graphical prunes	Zoom	Collaboration	Ontology Libraries
Apollo	Yes	Yes	No	Yes	Yes
LinKFactory	Yes	Yes	Yes	Yes	Yes
OntoStudio	No*	No	No*	Yes	Yes
Ontolingua Server	Yes	No	No	Yes	Yes
Ontosaurus	Yes	Yes	Yes	Yes	No
Protege	Yes	Yes	Yes	No	Yes

Table 6.4. Ontology editing tools: Usability

Tool	Graphical Taxonomy	Graphical prunes	Zoom	Collaboration	Ontology Libraries
WebODE	Yes	Yes	No	Yes	No
WebOnto	Yes	Yes	No	Yes	Yes
ICOM	Yes	No	No	No	Import and export as XML
IODE	No	No	No	Yes	Yes
Visual Ontology Modeler	Yes	No	No	Yes	Yes
Semtalk	Yes	No	No	Yes	Yes
COBra	Yes	No	No	No	Limited to GO and OBO
GKB	Yes	No	Yes	Yes	No
SWOOP	Yes	No	No	Yes	No
WSMT	Yes	No	Yes	No	No
WSMO Studio	No	No	No	No	Yes
Topbraid Composer	Yes	Yes	Yes	Yes	No
Neon Toolkit	Yes	Yes	Yes	Yes	No

Related to the usability of tools (Table 6.4), WebOnto has the most advanced features related to the cooperative and collaborative construction of ontologies. In general, more features are required in existing tools to ensure the successful collaborative building of ontologies. Finally, other usability aspects related to help system, editing and visualization should be improved in most of the tools.

6.2 Ontology Bootstrapping Approaches

There have been various approaches for semi-automatic generation of ontologies or taxonomies from underlying unstructured text data. These approaches can be broadly characterized as:

- Supervised-machine-learning based approaches, which require a large number of training examples, traditionally generated manually.
- Natural Language Processing (NLP) based approaches applied for generating ontological concepts and relationships. These are based on rules that analyze

patterns based on syntactic categories, which requires significant human involvement, making it expensive and infeasible for large-scale applications.

- Statistical clustering methods have been used to partition data-sets, categorize search results and visualize data. However, they have not focussed on generating labels for clusters and creation of new taxonomies.

Machine learning approaches are for the most part supervised, for which a set of manually generated positive and negative training examples are used. An approach using the concept-forming system COBWEB [118] has been used to perform incremental conceptual clustering on structured instances of concepts extracted from the Web [119]. Experimental and theoretical results on learning the CLASSIC description logic were presented in [120], and were used to construct concept hierarchies. An approach to bootstrapping a classification taxonomy based on a set of structured rules was proposed in [121]. A supervised approach presented in [122] supports semi-automatic and incremental bootstrapping of a domain-specific information extraction system.

Empirical and corpus-based NLP methods to build domain-specific lexicons have been proposed in [123] and used in [124]. Approaches that learn meanings of unknown words based on other word definitions in the surrounding context have been presented in [125] [126]. Case-based methods that match unknown word contexts against previously seen word contexts are described in [127] [128]. Approaches presented in [129] [130] apply shallow parsing, tagging and chunking, along with statistical techniques to extract terminologies or enhance existing ontologies. Full parse tree construction followed by decomposition into elementary dependency trees has been used to create medical ontologies from French text corpora in [131]. In [132], a thesaurus is built by performing clustering according to a similarity measure after having retrieved triples from a parsed corpus.

Linguistic structures such as verbs, appositions and nominal modifications have been used to identify hypernymic propositions in biomedical text [133]. Lexico-syntactic patterns have been investigated for inferring hyponymy from textual data in [134]. Salient words and phrases extracted from the documents are organized hierarchically using subsumption type co-occurrences in [130]. A description of supervised and unsupervised approaches to extract semantic relationships between terms in a text document is presented in [135]. A generalized association rule algorithm proposed in [136] detects non-taxonomic relationships between concepts and also determines the right level of abstraction at which to establish the relationship.

Effectively mining relevant information from a large volume of unstructured documents has received considerable attention in recent years [137] [138] [139]. A survey on the use of clustering in information retrieval is presented in [140]. Document clustering has been used for browsing large document collections in [141], using a “scatter/gather” methodology. These approaches create vector space representations of documents and use Euclidean or cosine-distance-based similarity metrics like the Euclidean ones to extract clusters from groups of documents. Clustering of Web documents to organize search results has been proposed in [142] [143].

There is a realization amongst researchers that one needs to leverage the strengths of a wide variety of techniques across machine learning, natural language processing and statistical approaches to address the difficult problem of ontology generation. Frameworks for hybrid approaches have been proposed:

- The ontology learning framework developed by Maedche and Staab [144].
- The Thematic Mapping System [144].
- The Taxaminer approach, which presents a framework to combine the techniques enumerated above [145].
- A complementary approach that uses the structure and content of HTML-based pages on the Web to generate ontologies [146].

6.3 Ontology Merge and Integration Tools

Ontology merging and integration, including functionalities related to versioning and keeping track of various changes, are very important in the context of the ontology design process. Furthermore, a large number of ontologies are being used to annotate content on the Web. There is a need to be able to reconcile annotations based on multiple ontologies and also support query processing across multiple ontologies. An approach to address the above challenge is to establish mappings between ontologies and to merge them at run time, as proposed in [147] [148]. A large number of ontology mapping and merging tools have appeared to address these issues. We now present a brief survey and a comparative evaluation of various ontology merge and integration tools [99]. The criteria used for the comparative evaluation of these tools are as follows:

Knowledge used during the merge process: The merging process can be much more efficient if additional knowledge can be made to bear on the process. Some examples of these knowledge resources are: electronic dictionaries, thesauri, lexicons, concept definitions and slot values, graph structures, instances of concepts and inputs from the user.

Interoperability: It is important because key activities such as transformation of formats and evaluation can be performed by other non-merging tools. Some important considerations are interoperability with other ontology tools or information systems and whether ontologies expressed in different languages can be merged.

Management of different versions of ontologies: A change in the source ontology results in a change in the merged ontology. Some important considerations are whether the tool takes advantage of the former versions of the ontologies and whether it warns that the merged ontology is not an accurate reflection of the source ontologies.

Components manipulated by the tools: An important consideration is about which components that can be merged by the ontology development tools or about which suggestions can be made by the merging tools. The main components that

need to be considered are concepts (including slots, and taxonomic and other relationships), axioms, rules and instances.

Editing and Visualization: This is very important for the usability of the tool. Some important considerations are support for a step-by-step view of the process, a simultaneous view of the source ontologies being merged, graphical prunes (views) of the ontologies being merged, zooming and the ability to hide/show information.

6.3.1 Ontology Merge and Integration Tools: A Brief Description

In this section, we present a brief description of some of the ontology merge and integration tools in use today.

Chimaera: Chimaera [149] is a merging and diagnostic Web-based browser ontology environment. It contains a simple editing environment in the tool and also allows the user to use the full Ontolingua editor/browser environment for more extensive editing. It facilitates merging by allowing users to upload existing ontologies into a new workspace (or into an existing ontology). Chimaera will suggest potential merging candidates based on a number of properties. Chimaera allows the user to choose the level of vigor with which it suggests merging candidates. Higher settings, for example, will look for things like possible acronym expansion. Chimaera also supports a taxonomy resolution mode. It looks for a number of syntactic term relationships, and when attached to a classifier, it can look for semantic subsumption relationships as well. Chimaera includes analysis capability that allows users to run a diagnostic suite of tests selectively or in its entirety. The tests include incompleteness tests, syntactic checks, taxonomic analysis, and semantic checks. Terms that are used but that are not defined, terms that have contradictory ranges, and cycles in ontology definitions are also detected.

PROMPT: PROMPT [150] is a tool for semi-automatic guided ontology merging, and is available as a plug-in for Protege. PROMPT leads the user through the ontology-merging process, identifying possible points of integration, and making suggestions regarding what operations should be done next, what conflicts need to be resolved, and how those conflicts can be resolved. PROMPT's ontology-merging process is interactive. A user makes many of the decisions, and PROMPT either performs additional actions automatically based on the user's choices or creates a new set of suggestions and identifies additional conflicts among the input ontologies. The tool takes into account different features in the source ontologies to make suggestions and to look for conflicts. These features include names of classes and slots, class hierarchy, slot attachment to classes, facets and facet values. Some conflicts identified by PROMPT are: name conflicts, dangling references, redundancy in the class hierarchy (more than one path from a class to a parent other than the root) and slot value restrictions that violate class inheritance.

ODEMerge: ODEMerge [151] is a tool to merge ontologies that is integrated in WebODE [107]. This tool is a partial software support for the methodology for merging ontologies [152], which proposes the following steps: (1) transformation

of formats of the ontologies to be merged; (2) evaluation of the ontologies; (3) merging of the ontologies; (4) evaluation of the result; and (5) transformation of the format of the resulting ontology to be adapted to the application where it will be used. WebODE helps in steps (1), (2), (4) and (5) of the merging methodology, and ODEMerge carries out the merge of taxonomies of concepts in step (3). Besides, ODEMerge helps in the merging of attributes and relations, and it incorporates many of the rules identified in the methodology. ODEMerge uses the source ontologies to be merged, and the synonymy, hyponymy and hypernymy relationships between terms across ontologies in the merging process. Customized dictionaries can be added to provide the relationships, and new merging rules can also be defined. ODEMerge supports the merging of ontologies in all the ontology languages supported by the WebODE tool.

6.3.2 Evaluation of Ontology Merge and Integration Tools

We now present a comparative evaluation of the various tools based on the dimensions identified earlier.

Table 6.5. Information used during the merge process

Feature	PROMPT	ODEMerge	Chimaera
Thesauri, Dictionaries	No	No	No
Lexicons	No	No	No
Concept Definitions and Slot Values	Yes	Yes	Yes
Graph Structure	Yes	Yes	No
Instances of concepts	Yes	No	No
User Input	Yes	Yes	Yes

Table 6.5 compares the information used by these tools (electronic dictionaries, lexicons, etc.) during the merge process. The more information a tool uses during this process, the more work it is able to perform without the user's participation. Most of the tools start the merging process by searching for similar concepts of ontologies.

Table 6.6. Interoperability

Feature	PROMPT	ODEMerge	Chimaera
Tools and systems Interoperability	Yes	Yes	Yes
Multiple ontology language support	Yes	Yes	Yes

Interoperability with other ontology tools is also an important aspect (Table 6.6) and is usually determined by the ontology development platform in which the merge tool is integrated. Another important aspect is whether the tool can merge ontologies expressed in different languages. All these tools are able to merge ontologies expressed in different languages (XML, RDFS, OIL, etc.).

Table 6.7. Management of different versions

Feature	PROMPT	ODEMerge	Chimaera
Leveraging previous ontology versions	No	No	No
Notifications of changes in source ontologies	No	No	No

Given that ontologies usually evolve, the management of different ontology versions is also important (Table 6.7). None of these tools takes advantage of former versions of the ontologies to be merged, and none of them warns users about changes in the source ontologies.

Table 6.8. Components of ontologies manipulated by the tools

Feature	PROMPT	ODEMerge	Chimaera
Concepts	Merge, Suggest	Merge	Merge, Suggest
Own Slots	Merge, Suggest	Merge	Merge, Suggest
Template Slots	Merge, Suggest	Merge	Merge, Suggest
Taxonomies	Merge, Suggest	Merge	Merge, Suggest
Concepts	Merge, Suggest	Merge	Merge, Suggest
Relations	Merge	Merge	Not currently supported
Partitions and/or Decompositions	No	Merge	Both supported
Relations or Functions?	Merge, Suggest (relations only)	Merge	Not currently supported
Arity	Merge, Suggest (binary relations)	Merge	No
Sets of axioms	No	No	No
Sets of rules	No	No	No
Instances	Merge, Suggest	No	Not currently supported
of Concepts	Merge, Suggest	No	No

Table 6.8. Components of ontologies manipulated by the tools

Feature	PROMPT	ODEMerge	Chimaera
of Relations	Merge, Suggest	No	No
Claims	No	No	No

We present in Table 6.8 which kinds of components can be merged by the tool and about which kinds of component merging suggestions can be proposed by the tools. All the tools allow merging concepts, taxonomies, relations and instances. However, no tool allows merging axioms and rules. From among all of them, PROMPT is the tool that provides most suggestions to the users.

Table 6.9. Editing and visualization support

Feature	PROMPT	ODEMerge	Chimaera
Step by Step view of Process	Graphical, tabular, hierarchical	Non-graphical	HTML text
Simultaneous view of source ontologies	Yes	No	Yes
Graphical view of source ontologies	Through host tool	Through host tool	No
Zoom	Through host tool	Through host tool	No
Hide/Show	Through host tool	Through host tool	Yes: for subclass/superclass relationships and child slots

Edition and visualization features (Table 6.9) are strongly influenced by the ontology development platform in which these tools are integrated.

6.4 Ontology Engines and Reasoners

In this section, we present a brief description of ontology reasoners and engines. Whereas most of the ontology-reasoning systems are based on description logics, some reasoners are implemented based on rules and first-order logic theorem provers.

CEL: The system CEL [153] is a description logic system that offers both sound and complete polynomial-time algorithms and expressive means that allow its use in real-world applications. It is based on recent theoretical advances that have shown that the description logics (DL) EL, which allows for conjunction and existential restrictions, and some of its extensions have a polynomial-time subsumption problem even in the presence of concept definitions and so-called general concept

inclusions (GCI). The DL EL+ handled by CEL extends EL by so-called role inclusions (RI). On the practical side, it has turned out that the expressive power of EL+ is sufficient to express several large life science ontologies. In particular, the Systematized Nomenclature of Medicine (SNOMED) [7] can be expressed using EL with RIs and acyclic concept definitions. The Gene Ontology (GO) [10] can also be expressed in EL with acyclic concept definitions and one transitive role (which is a special case of an RI). Finally, large parts of the Galen Medical Knowledge Base (GALEN) [154] can be expressed in EL with GCIs and RIs.

CEL is a tractable fragment of OWL 1.1 [407], which is an extension of OWL and is currently a W3C member submission. A W3C group is currently working on creating the next version of the OWL, to be christened OWL 2.0, based on this submission. This tractability is achieved by eliminating the `allValuesFrom` construct and retaining the `someValuesFrom` construct. CEL also supports the role inclusion axioms e.g., `hasStructuredTestResult o indicatesDisease => suffersFrom`. The constructs from OWL 1.1 which cause intractability are cardinality restrictions, union, negation, inverse properties, functional and inverse functional properties.

FaCT++: FaCT++ [155] is the new generation of the well-known FaCT [156] OWL-DL reasoner. FaCT++ uses the established FaCT algorithms, with a different internal architecture, and is implemented using C++ for efficiency and portability. Some interesting features of FaCT are: (a) its expressive logic (in particular the SHIQ reasoner): SHIQ is sufficiently expressive to be used as a reasoner for the DLR logic, and hence to reason with database schemas; (b) its support for reasoning with arbitrary knowledge bases (i.e., those containing general concept inclusion axioms); (c) its optimized tableaux implementation (which has now become the standard for DL systems); and (d) its CORBA- based client server architecture.

fuzzyDL: fuzzyDL [157] is a Description Logics Reasoner supporting Fuzzy Logic reasoning. The fuzzyDL system includes a reasoner for fuzzy SHIF with concrete fuzzy concepts (ALC augmented with transitive roles, a role hierarchy, inverse roles, functional roles, and explicit definition of fuzzy sets). Some interesting features of fuzzyDL are: (a) extension of the classical Description Logics SHIF to the fuzzy case; (b) explicit definitions of fuzzy concepts with left-shoulder, right-shoulder, triangular and trapezoidal membership functions; (c) concept modifiers in terms of linear hedges; (d) support for General Inclusion Axioms; (e) support for "Zadeh semantics" and Lukasiewicz logic; and (f) backward compatibility, i.e. it support for classical description logic reasoning.

KAON2: KAON2 [158] is an infrastructure for managing OWL-DL, SWRL, and F-Logic ontologies and has the following interesting features: (a) an API for programmatic management of OWL-DL, SWRL, and F-Logic ontologies; (b) a stand alone server providing access to ontologies in a distributed manner using RMI; (c) an inference engine for answering conjunctive queries (expressed using SPARQL syntax); (d) a DIG interface, allowing access from tools such as Protege; and (e) a module for extracting ontology instances from relational databases.

Pellet: Pellet [159] is an open source OWL-DL reasoner written in Java, originally developed at the University of Maryland's Mindswap Lab, and funded by a diverse group of organizations. Pellet is based on the tableaux algorithms developed for expressive Description Logics (DLs). It supports the full expressiveness of OWL-DL including reasoning about nominals (enumerated classes). In addition to OWL-DL, as of version 1.4, Pellet supports all the features proposed in OWL 1.1, with the exception of n-ary datatypes. Thus the expressiveness of supported DL is SROIQ(D), which extends the well-known DL SHOIN(D) (the DL that corresponds to OWL-DL) with qualified cardinality restrictions, complex subproperty axioms (between a property and a property chain), local reflexivity restrictions, reflexive, irreflexive, symmetric, and anti-symmetric properties, disjoint properties, and user-defined datatypes. Pellet provides many different reasoning services such as consistency checking, concept satisfiability, classification and realization. It also incorporates various optimization techniques described in the DL literature and contains several novel optimizations for nominals, conjunctive query answering and incremental reasoning.

RacerPro: RacerPro [160] provides a first implementation of the Semantic Web Rules Language (SWRL) in its latest version. It also supports services for OWL ontologies and RDF data descriptions such as: (a) consistency checking for OWL ontologies and a set of data descriptions; (b) inference of implicit subclasses and synonyms for resources (classes or instances); (c) OWL-QL query processing for OWL documents (ontologies and their instances); and (d) incremental query answering for information retrieval tasks. RacerPro implements a highly optimized tableau calculus for a very expressive description logics. It offers reasoning services for multiple T-boxes and for multiple A-boxes as well. The system implements the description logic $ALCQHI_R$, also known as SHIQ. This is the basic logic ALC augmented with qualifying number restrictions, role hierarchies, inverse roles, and transitive roles. In addition to these basic features, RacerPro also provides facilities for algebraic reasoning including concrete domains for dealing with min/max restrictions over the integers; linear polynomial (in)equations over the reals or cardinals with order relations; and equalities and inequalities of strings. RacerPro combines description logic reasoning with, for instance, reasoning about spatial (or temporal) relations within the A-box query language nRQL. Bindings for query variables that are determined by A-box reasoning can be further tested with respect to an associated constraint network of spatial (or temporal) relationships.

Jena: The Jena2 inference subsystem [408], is designed to allow a range of inference engines or reasoners to be plugged into Jena. Such engines are used to derive additional RDF assertions which are entailed from some base RDF together with any optional ontology information and the axioms and rules associated with the reasoner. The primary use of this mechanism is to support the use of languages such as RDFS and OWL which allow additional facts to be inferred from instance data and class descriptions. However, the machinery is designed to be quite general and, in particular, it includes a generic rules engine that can be used for many RDF

processing or transformation tasks. Other than the generic rules engine, there are other pre-defined reasoners included in the Jena2 system, such as a transitive reasoner that stores and traverses class and property lattices, a RDFS rule reasoner that implements a configurable set of RDFS entailments, and a set of reasoners for various subsets of OWL.

JESS: Jess [409], is a rules engine and scripting environment written entirely in Sun's Java language at Sandia National Laboratories in Livermore, CA. Using Jess, it is possible to build Java software that has the capacity to perform reasoning using knowledge supplied in the form of declarative rules. Jess includes a full-featured development environment based on the award-winning Eclipse platform. Jess uses an enhanced version of the Rete algorithm [410] to process rules. Jess has many unique features including backward chaining and working memory queries, and Jess can directly manipulate and reason about Java objects.

6.5 Clinical Scenario Revisited

Consider the clinical use case scenario presented in Chapter 2. In particular consider the important issue of knowledge change propagation in this section. Consider the definition in natural language of fibric acid contraindication:

```
A patient is contraindicated for fibric acid if he or she has an
allergy to fibric acid or has an abnormal liver panel.
```

Suppose there is a new (hypothetical) biomarker for fibric acid contraindication for which a new molecular diagnostic test is introduced in the market. This leads to a redefinition of a fibric acid contraindication as follows.

```
The patient is contraindicated for fibric acid if he has an allergy to
fibric acid or has elevated Liver Panel or has a genetic mutation.
```

Let us also assume that there is a change in a clinically normal range of values for the lab test AST which is a part of the liver panel lab test. This leads to a knowledge change and propagation across various knowledge objects that are sub-components and associated with the fibric acid contraindication concept. A diagrammatic representation of the OWL representation of the new fibric contraindication with the changes marked in red ovals is illustrated below. The definition of "fibric acid contraindication" changes, triggered by changes at various levels of granularity.

A potential sequence of change propagation steps are enumerated below:

1. The clinically normal range of values for the AST lab result changes.
2. This leads to a change in the abnormal value ranges for the AST lab result
3. This leads to a change in the definition of an abnormal liver panel.

4. This leads to a change in what it means to be a patient with an abnormal liver panel.
5. The definition of fibric acid contraindication changes due to the following changes:
   (A) The change in the definition of a patient with an abnormal liver panel as enumerated in steps 1-4 above.
   (B) Introduction of a new condition: a patient having a mutation: "Missense: XYZ3@&%" (hypothetical). This is a new condition which could lead to a change in what it means to be a patient with a contraindication to fibric acid.

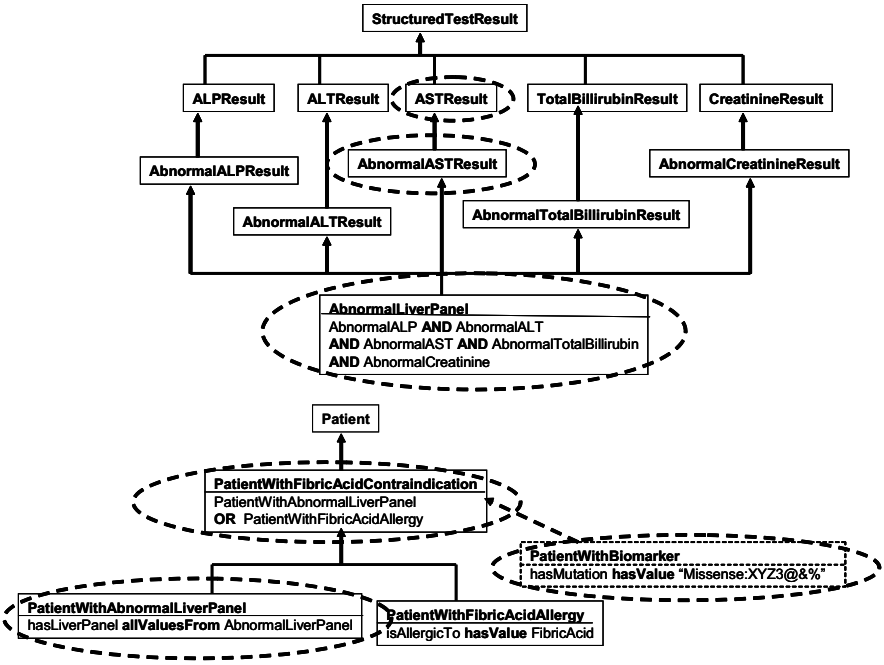

Fig. 6.1. Knowledge change and propagation

It may be noted that in our discussion in Section 6.3.1, none of the ontology editors and tools today support versioning and change management functionality. In our solution approach, we propose to load these ontologies as data into a rules engine and write specialized rules to identify the impacts of a change operation.

6.6 Summary

In this chapter, we presented a discussion on different aspects of ontology authoring, bootstrapping and management. In particular, we present a survey of ontology building tools, ontology-reasoning engines, and techniques for ontology bootstrap-

ping, matching, merging and integration. A more detailed account of ontology authoring and management may be obtained from the Handbook on Ontologies by Staab and Ruder [411]. The clinical use case is revisited and an approach for modeling knowledge change and propagation as ontology versioning and change management is presented.