

## CHAPTER 4



# Semantic Web Development Tools

Extracting and manipulating RDF from semistructured data and writing client applications to handle RDF data are common tasks that can be made easier and more efficient using software tools. Web designers and search engine optimization (SEO) experts often generate machine-readable annotations or convert existing structured data to a different serialization. While web site markup can be edited in any text editor, some advanced features are desired when working with semantic annotations, so an advanced text editor is a fundamental tool. Annotators can be used for semantically annotating your web pages and RDFizers to convert text, HTML, and XML documents to RDF. Application developers writing Semantic Web applications in Java, JRuby, Clojure, Scala, Python, and other programming languages often work with Integrated Development Environments, many of which support the integration of semantic software libraries. Ontology editors are widely deployed and used in ontology engineering, many of which support reasoning as well. Linked Data software tools are useful for extracting Linked Data, visualizing Linked Data interconnections, as well as exporting and publishing Linked Data. Semantic Web browsers can display structured data extracted from web pages, generate a map from geospatial data on your smartphone, and provide advanced navigation and interactivity features unavailable in traditional web browsers.

## Advanced Text Editors

In contrast to word processors such as Microsoft Word or OpenOffice.org Writer, *plain-text editors* cannot be used for document formatting, but they are suitable for creating and modifying web pages. However, basic text editors are not convenient for web design, because some vital features are missing from them. For example, many of them do not handle control characters and whitespaces correctly. The most well-known examples are Notepad under Windows and vi under Linux. *Advanced text editors* such as WordPad provide text formatting and other additional features. Some advanced text editors are also source code editors with additional tools specifically designed for web designers and software engineers. While not suitable for structured data conversions or LOD processing, advanced text editors are fundamental programs in the toolbox of every Semantic Web developer, owing to advanced features such as the following:

- Comprehensive character encoding support, including full Unicode support
- Whitespace character support
- Control character support, for example, CR+LF (Windows), LF only (UNIX), and Apple (CR only) break rows
- Multifile editing with tabs

- Customizable color schemas for *syntax highlighting* (HTML, CSS, XML,<sup>1</sup> scripts, and so on)
- Undo/redo
- Forced word wrap
- Line numbering
- Auto-indent
- Guides for tag pairs and element nesting
- OS integration (adds application to right-click menu)

The selected editor should be integrated with at least one of your browsers as the default source code editor, which you can use to open the currently rendered web document with a hot key (usually Ctrl+U). There are additional features of text editors that are not vital but can be useful.

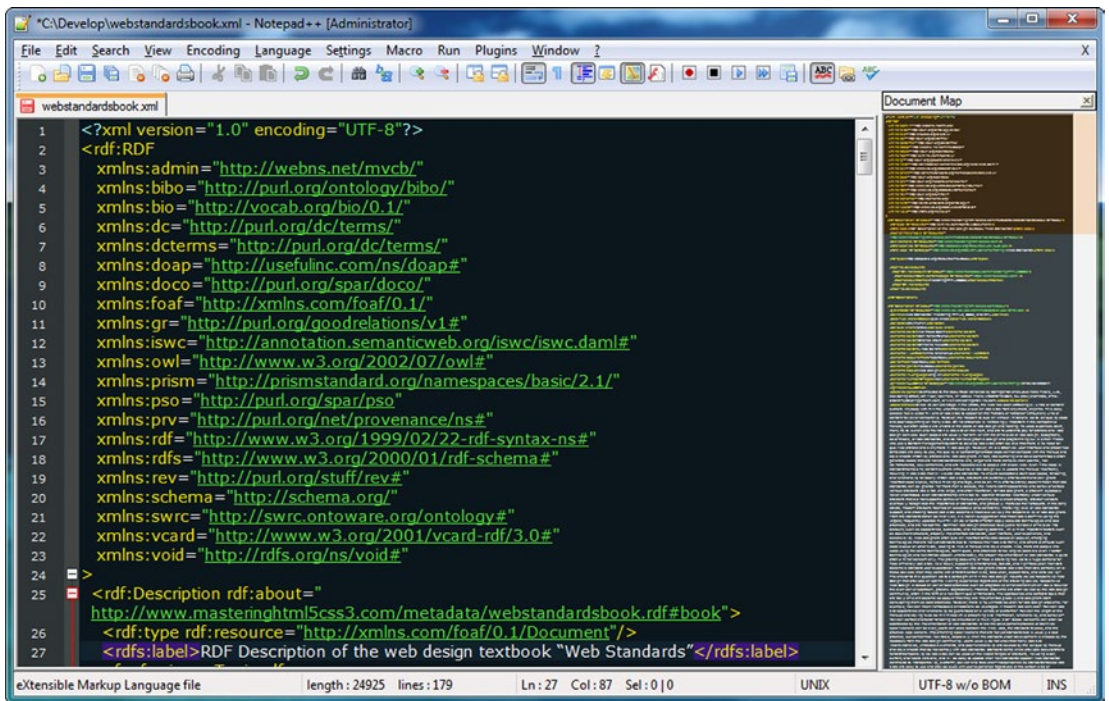
- Customized color and font settings
- Customizable toolbars
- Spell checker
- Templates
- Bookmarks
- Full drag-and-drop support
- Built-in FTP client or integration with an (S)FTP client
- Conversions (uppercase, lowercase, invert case, and initial caps)
- International versions (can be convenient for some developers)
- Support for double-byte character systems (DBCS) used in Far East Asian languages, such as Chinese or Japanese (if required)
- Browser preview (launching the default or selected web browser for debugging and testing)

Some of the most well-known advanced text editors are EditPlus and Notepad++ (free, open source [1]) for Windows, BlueFish [2] and Komodo Edit [3] for Linux, and BBEdit [4] and TextWrangler [5] for Mac OS. A comprehensive cross-platform editor is Arachnophilia, which is available for Windows, Linux, Unix, FreeBSD, and Mac OS [6].

As an example, let's look at the major features of Notepad++ . It is a multifile editor with convenient file manager options. Notepad++ saves multiple files with a single click, opens recently edited files, and provides tabs for each opened file. It has a fully customizable interface with advanced features such as line markers, guides for opening and closing tag pairs, structuring guides to collapse or reveal the currently edited level of the DOM tree, and syntax highlighting (see Figure 4-1).

---

<sup>1</sup>On Windows systems, the file format used for syntax highlighting depends on the file extension, so an entire RDF/XML file with the `.rdf` extension might be white by default, while the same file in the same editor would be syntax-highlighted when saved as `.xml`.



**Figure 4-1.** Syntax highlighting and tag pair guides in Notepad++

There is a variety of programming and web development languages supported in syntax highlighting, from HTML to XML and from PHP to Ruby. There are several predefined color themes you can select from, or you can create new ones to your taste. The different document components (indent guidelines, marks, carets, whitespaces, tag pairs, active and inactive tabs, and so on) can be styled individually. Notepad++ can change text direction of documents. It also supports a variety of character encodings, can add and remove byte-order marks, supports big-endian and little-endian Unicode files, and converts files from one encoding to another.<sup>2</sup> The documents opened in the application can be previewed in any installed browsers.

Notepad++ also provides advanced text transformation functionalities, such as escaping certain characters, transforming lowercase characters to uppercase (or vice versa), searching for matching strings, converting decimal numbers to their hexadecimal equivalents, inserting the current date and time, sorting lists ascending or descending, automatically converting leading spaces to tabs, and so on. Notepad++ also supports macros, which you can run multiple times. The list of features can be extended through additional plug-ins, such as the MIME tools for Base64 encoding and decoding.

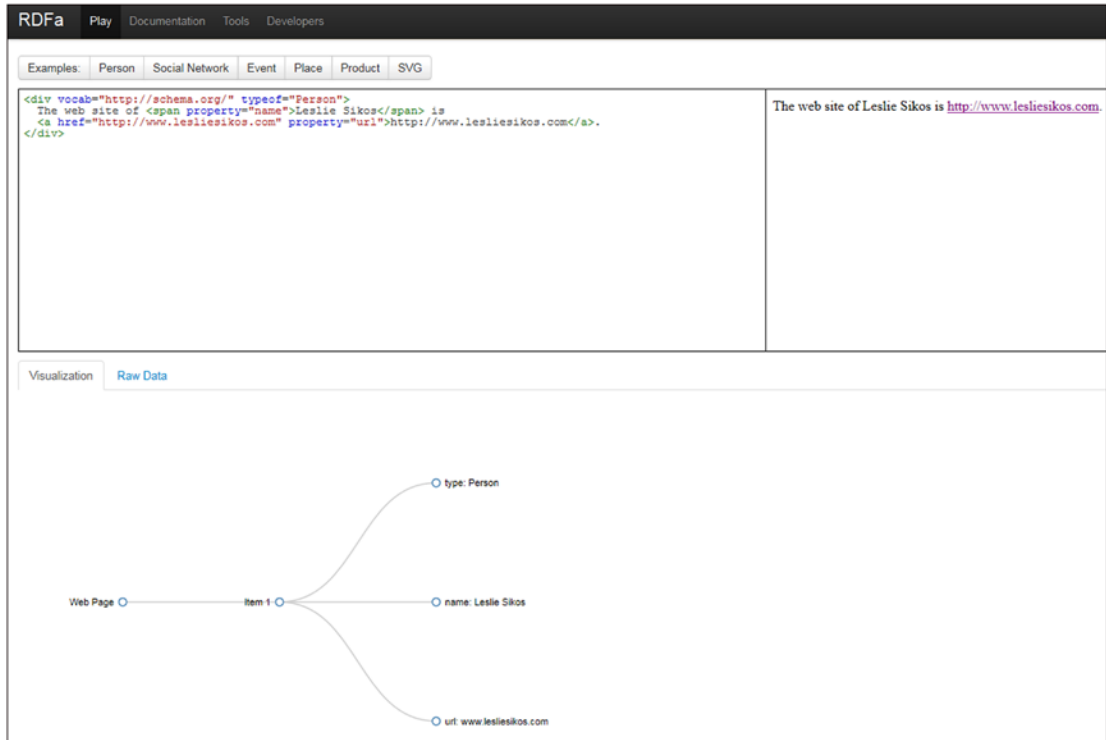
## Semantic Annotators and Converters

While there are templates available for all machine-readable metadata annotations and one might also write them manually from scratch, you can use software tools that can evaluate your code, provide a preview of the human-readable part of your markup, as well as extract RDF triples, generate the RDF graph of your structured data, and/or convert the annotation to other formats, which can be very handy, owing to the large number of RDF serializations.

<sup>2</sup>This feature should be used for those encodings that can be reasonably converted to another, more advanced encoding without sacrificing special characters (for example, ANSI to UTF-8).

## RDFa Play

*RDFa Play* is a real-time RDFa 1.1 editor, data visualizer, and debugger available at <http://rdfa.info/play/>. It takes the raw RDFa input, generates a live preview for the human-readable data, and generates a graph from the triples (see Figure 4-2). If you modify the code, RDFa Play regenerates the browser preview and the graph.



**Figure 4-2.** Live browser preview and graph in RDFa Play

RDFa Play provides RDFa annotation examples for persons, events, and places using schema.org, personal data expressed in FOAF, product description in GoodRelations, and Dublin Core metadata in SVG.

## RDFa 1.1 Distiller and Parser

W3C's *RDFa 1.1 Distiller and Parser* at <http://www.w3.org/2012/pyRdfa/> processes your HTML markup containing RDFa and converts the triples to Turtle, RDF/XML, JSON-LD, or N-Triples. The RDFa 1.1 Distiller and Parser is written in Python and powered by RDFLib (<https://rdflib.readthedocs.org>). It accepts online RDFa code fragments, uploaded files, and RDFa annotations copied and pasted. The supported host languages for file upload and direct input are HTML5+RDFa, XHTML+RDFa, SVG+RDFa, Atom+RDFa, and XML+RDFa.

## RDF Distiller

The *RDF Distiller* at <http://rdf.greggkellogg.net/distiller> integrates RDF graphs, readers, and writers to Ruby projects. The distiller can be used to transform data between different RDF serializations. The web interface provides a form, which takes the user input through a URI or as direct input in JSON, JSON-LD, HTML5 Microdata, N3, N-Quads, N-Triples, RDFa, RDF/XML, TRiG, TRiX, or Turtle, and converts the code to any of the formats (see Figure 4-3).

From URL
From Form Input

---

Distill RDF Information

URI

---

Options

Input Format:  Output Format:

Show Parser debug information:

Validate Input:

Expand graph (RDFa only):

Don't Verify SSL:

Output Graph (RDFa only):

Raw Output:

---

Markup

```

<div vocab="http://schema.org/" typeof="Person">
  <span property="name">Leslie Sikos</span>
  
  Leslie's web site:
  <a href="http://www.lesliesikos.com" property="url">lesliesikos.com</a>
</div>

```

## Result

Note, to see result directly, such as formatted HTML for RDFa output, choose the 'Raw Output' option above.

```

@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/ns/rdfs#> .
@prefix schema: <http://schema.org/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<> rdfs:usesVocabulary schema: .

[
  a schema:Person;
  schema:image <lesliesikos.jpg>;
  schema:name "Leslie Sikos";
  schema:url <http://www.lesliesikos.com>
] .

```

**Figure 4-3.** RDFa to Turtle conversion in RDF Distiller

The Distiller can automatically detect the input format, which can also be explicitly selected from a drop-down list.

## DBpedia Spotlight

DBpedia Spotlight is a tool for annotating DBpedia concepts in plain text [7]. It has three basic functions: annotation, disambiguation, and marking candidates. DBpedia Spotlight’s web application visualizes the user’s input with DBpedia resource annotations (see Figure 4-4).



**Figure 4-4.** Annotation with DBpedia Spotlight

The RESTful, SOAP-based web API exposes the functionality of annotating and disambiguating entities. The annotation Java/Scala API exposes the underlying logic that performs the annotation or disambiguation. The indexing Java/Scala API executes the data processing necessary to enable the annotation or disambiguation algorithms used.

## Google Structured Data Testing Tool

The *Google Structured Data Testing Tool* at <http://www.google.com/webmasters/tools/richsnippets> is suitable for machine-readable metadata testing, including Microformats, RDFa, and HTML5 Microdata annotations online or through direct input. The code length of the direct input is limited to 1,500 characters. The tool provides a preview of Google’s representation of your site on Search Engine Result Pages (SERPs), along with the extracted structured data as item, type, and properties (see Figure 4-5).

Extracted structured data

Item	
type:	http://schema.org/person
property:	
name:	Leslie Sikos
memberof:	W3C

**Figure 4-5.** Triples extracted by the Google Structured Data Testing Tool

The tool can identify incomplete triples and provides a short explanation if any mandatory property is missing. The Google Structured Data Testing Tool also indicates properties that are not parts of the vocabulary used for the object.

---

■ **Note** Google does not use machine-readable metadata annotations on Search Engine Result Pages if certain properties are missing for a particular object type. For example, an hCard description will be used by Google only if you provide not only the name but also at least two of the following three properties: organization, location, or role, while code validity can be achieved even if you omit them.

---

The tool provides machine-readable metadata examples for applications, authors, events, music, people, products, product offers, recipes, and reviews; however, you must log in to your Google account to retrieve the HTML markup of the examples. All other functionalities are available without logging in.

## RDFizers

Those software tools that convert application and web site data to RDF are called *RDFizers*. They can be used for a one-time migration effort or implemented as middleware components of Semantic Web software tools such as OpenLink Data Explorer. RDFizers are often available as a software library.

### Apache Any23

Apache *Anything To Triples* (Any23) is a Java library, RESTful web service, and command-line tool available at <https://any23.apache.org>. Any23 extracts structured data from a variety of Web documents, including RDF serializations such as RDF/XML, Turtle, Notation 3, and RDFa; Microformats such as Adr, Geo, hCalendar, hCard, hListing, hRecipe, hReview, License, XFN and Species; HTML5 Microdata; JSON-LD; CSV (Comma Separated Values exported from, for example, Microsoft Excel); as well as vocabularies such as Dublin Core, DOAP, FOAF, GeoNames, Open Graph Protocol, schema.org, and vCard. Any23 can also be used for data conversion such as Turtle to N-Triples.

Apache Any23 can perform validation for code quality assurance. It automatically fixes the DOM structure if it detects incorrect HTML element nesting. Any23 can identify not only structuring markup elements but also meta tags and RDFa annotations. If, for example, a prefix mapping is missing for an RDFa annotation, the RDFa parser will find it out of context and will not be able to handle it. To address this, Apache Any23 provides the *Validator* classes to implement a Rule precondition, which, when matched, will trigger the *Fix* method to correct the code.

Owing to its comprehensive features, Any23 is implemented in major Semantic Web applications, such as Sindice.



## General Architecture for Text Engineering (GATE)

The *General Architecture for Text Engineering (GATE)*, an open source text processor tool developed by the University of Sheffield, uses Natural Language Processing (NLP) methods to generate RDF from text files [8]. GATE's Ontology plug-in provides an API for manipulating OWL-Lite ontologies that can be serialized as RDF and RDFS. If you work with OWL-DL ontologies, classes that are subclasses of restrictions supported in OWL-Lite are usually shown, but the classes that are subclasses of other restrictions will not be displayed. Similarly, plain RDF/RDFS files will not be shown correctly, because there is no way for the API to represent many constructs that are allowed in RDF but not allowed in OWL-Lite.

## OpenRefine

*OpenRefine* is a tool for exploring large datasets, cleaning and transforming data from one format to the other, reconciling and matching data, extending data with web services, and linking data to LOD databases [9]. With OpenRefine, you can filter and partition data with regular expressions, use named-entity extraction on full-text fields to automatically identify topics, and perform advanced data operations with the General Refine Expression Language.

## Ontology Editors

Ontology editors are software tools specifically designed for ontology engineering. They cover the common tasks of all major stages of ontology development, namely they

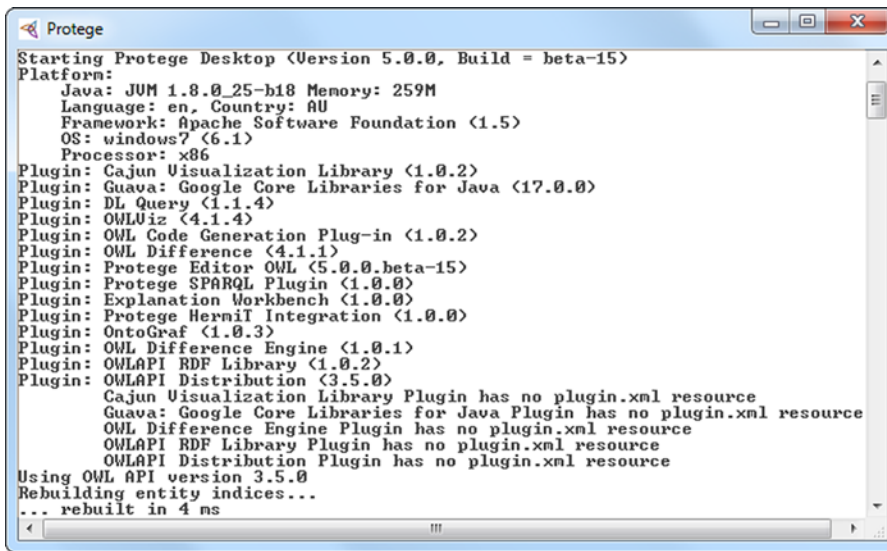
- Determine domain and scope. What is the knowledge domain the ontology will cover? What are the potential implementation areas? What types of questions does it intend to answer?
- Consider reuse. Assess other ontologies of similar knowledge domains.
- Enumerate important terms. Create a comprehensive list of terms for the chosen knowledge domain, without focusing on class hierarchy, properties, overlapping terms, or relationships.
- Define classes and class hierarchy.
- Define properties, and characteristics of properties. Define property types, including simple properties and relationships to classes, domains and ranges, as well as universal, existential, and cardinality restrictions.
- Create individuals.

## Protégé

Stanford University's *Protégé* is the most widely used open source ontology editor and knowledge management toolset, which can be downloaded from <http://protege.stanford.edu>. It supports reasoners such as HermiT and FaCT++ to validate ontologies for consistency, as well as a variety of other plug-ins. Originally developed as a Learning Health System for translating raw biomedical data into machine-readable data for decision making, Protégé is now suitable for modeling, ontology-driven application development, and collaborative ontology engineering. The ontologies can be exported in many formats, such as RDFS, and various OWL syntaxes.

While the ontologies can be created in Protégé through a Graphical User Interface (GUI), the software is Java-based, so when it is executed, it opens a command line (see Figure 4-6) behind the GUI in a separate window. The ontologies created in Protégé can be accessed from Java programs through the Protégé-OWL API.





```

Starting Protege Desktop <Version 5.0.0, Build = beta-15>
Platform:
  Java: JUM 1.8.0_25-b18 Memory: 259M
  Language: en, Country: AU
  Framework: Apache Software Foundation <1.5>
  OS: windows? <6.1>
  Processor: x86
Plugin: Cajun Visualization Library <1.0.2>
Plugin: Guava: Google Core Libraries for Java <17.0.0>
Plugin: DL Query <1.1.4>
Plugin: OWLviz <4.1.4>
Plugin: OWL Code Generation Plug-in <1.0.2>
Plugin: OWL Difference <4.1.1>
Plugin: Protege Editor OWL <5.0.0.beta-15>
Plugin: Protege SPARQL Plugin <1.0.0>
Plugin: Explanation Workbench <1.0.0>
Plugin: Protege Hermit Integration <1.0.0>
Plugin: OntoGraf <1.0.3>
Plugin: OWL Difference Engine <1.0.1>
Plugin: OWLAPI RDF Library <1.0.2>
Plugin: OWLAPI Distribution <3.5.0>
Cajun Visualization Library Plugin has no plugin.xml resource
Guava: Google Core Libraries for Java Plugin has no plugin.xml resource
OWL Difference Engine Plugin has no plugin.xml resource
OWLAPI RDF Library Plugin has no plugin.xml resource
OWLAPI Distribution Plugin has no plugin.xml resource
Using OWL API version 3.5.0
Rebuilding entity indices...
... rebuilt in 4 ms

```

Figure 4-6. Protégé's command line

The GUI of Protégé features a main menu, an address bar, and a tab-based editor (see Figure 4-7).

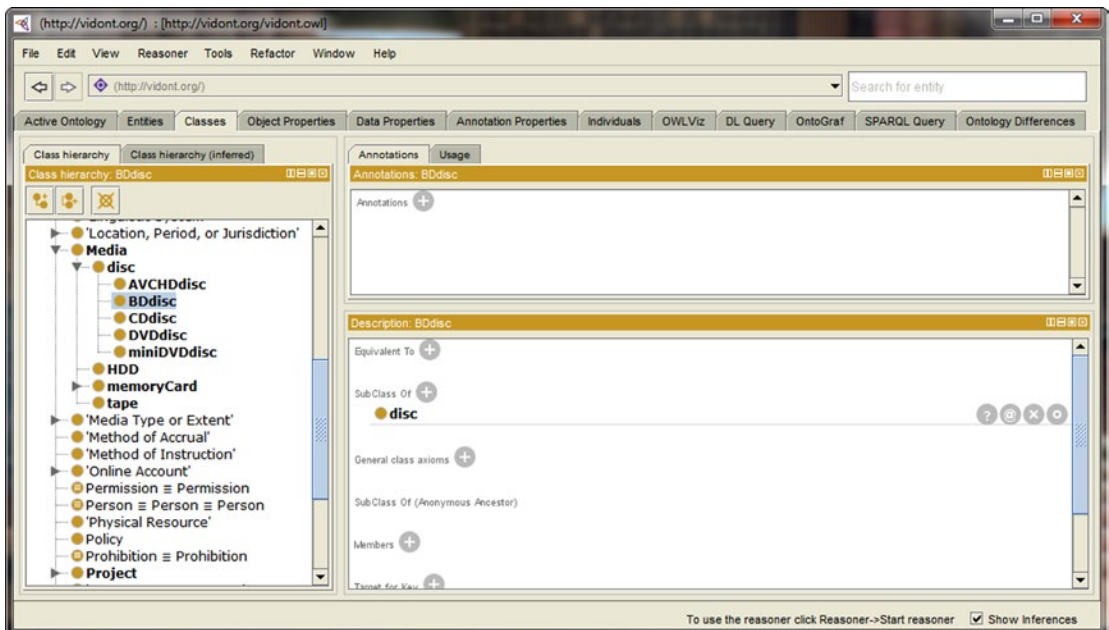


Figure 4-7. Protégé's Graphical User Interface

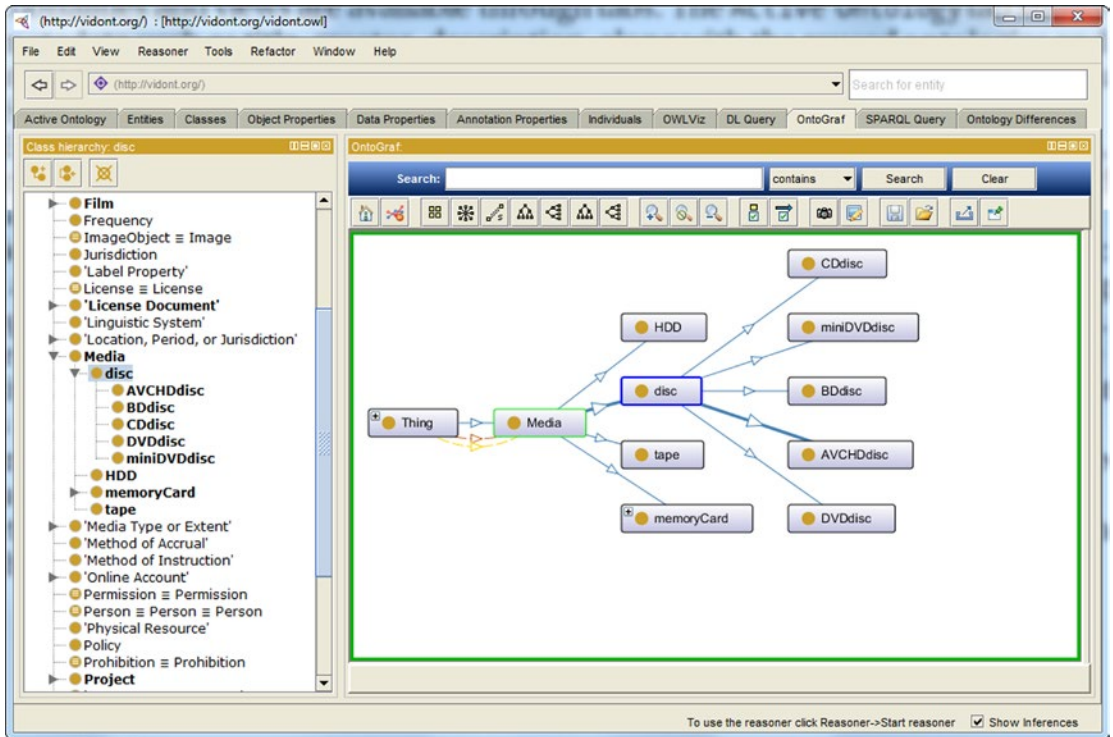
In the File menu, you can create a new, empty ontology or open an ontology from an offline or online .owl file. Ontologies can be saved in a variety of formats, including RDF/XML, OWL/XML, OWL Functional Syntax, Manchester Syntax, OBO (Open Biomedical Ontologies format), KRSS2 (Knowledge Representation System Specification v2), Latex, or Turtle. The wide range of plug-ins available for Protégé can be downloaded and the already installed plug-ins updated also from this menu.

Under File ► Preferences, you can handle hidden annotation URIs. To make it easier to automatically generate unique identifiers to classes, properties, and individuals of an ontology, you can set up or modify the structure of entity URIs for a particular ontology. Once you set up the base URI of the ontology, all fragment identifiers of the ontology will start with this address, which can be modified any time later (New Ontologies tab in File ► Preferences). This can be very useful if the address structure has to be changed after creating the ontology, because the developer does not have to change the hundreds or thousands of addresses manually one by one. The default base URI can be a web address of your choice, and the path can optionally include the actual year, month, and day. The base URI typically ends in a #, but this can be changed to / or :, if needed (New Entities tab in File ► Preferences). The number sign is the default setting, however, because it creates valid fragment identifiers. You can set the ending of the entity URIs to an arbitrary name, which is the default choice. If you want to use automatically generated identifiers instead, you can set entity labels, including custom URIs and a globally unique prefix or suffix.

OWLviz, a Protégé plug-in installed by default, powers the graphical representation of class hierarchies of OWL ontologies and the navigation between the classes represented as a tree structure (OWLviz tab in File ► Preferences). OWLviz makes the comparison of the asserted class hierarchy and the inferred class hierarchy possible. By default, Protégé automatically checks for plug-in updates at program startup, which can also be disabled (Plugins tab in File ► Preferences). The default plug-in repository is set to GitHub, which can be changed. The Reasoner tab in File ► Preferences can display or hide class, object property, data property, and object inferences or initialize reasoners by setting up precomputation tasks such as classification or realization to be done when the reasoner is launched. The tree hierarchy can be automatically expanded under Tree Preferences in File ► Preferences by setting an auto-expansion depth limit (the default value is 3) and an auto-expansion child count limit (the default value is 50). By default, automatic tree expansion is disabled. Accidentally performed changes on any tab can be reverted by clicking the Reset preferences... button on the bottom right-hand corner of File ► Preferences.

The core functionalities and views are available through tabs. The Active Ontology tab shows general ontology metadata, such as title, creator, description, in addition to the reused ontologies and statistics about ontology metrics, such as the number of axioms, classes, object properties, individuals, and so on. Protégé also displays all the prefixes used in the opened ontology. Protégé features a dedicated tab for Entities, Classes, Object Properties, Data Properties, Annotation Properties, and Individuals. The class hierarchy is shown as a tree structure, wherein each node can be opened or closed individually. The selected entity, class, or property details are shown in separate panels. Class descriptions provide information about equivalent classes, subclasses, class axioms, members, etc., of the selected class, as well as the option to change the values or add new ones. The classes in Protégé are subclasses of Thing and overlap by default. Class hierarchies can be created from the Tools menu. The object or data type properties can have subproperties or inverse properties. The properties can be functional, transitive, symmetric, asymmetric, reflexive, or irreflexive. Protégé automatically updates inverse properties (such as hasChild and isSonOf in a family relationship ontology).

The Object Properties and Data Properties tabs also have a Characteristics panel. For object properties, the Characteristics panel features checkboxes for Functional, Inverse functional, Transitive, Symmetric, Asymmetric, Reflexive, and Irreflexive properties. The Individuals tab shows not only the class hierarchy but also the members list and the property assertions. The OntoGraf tab provides a visual representation of any part of the ontology (see Figure 4-8). When you hover the mouse over any part of the graph, Protégé shows the fragment identifier, as well as the subclasses/superclasses (if any).



**Figure 4-8.** Graph visualization in Protégé

The SPARQL Query tab provides an interface to execute SPARQL queries. Protégé enumerates the prefixes, provides an editable SELECT query template, which you can modify or delete, and adds arbitrary queries.

Protégé also has an online version at <http://webprotege.stanford.edu>, which has collaboration support.

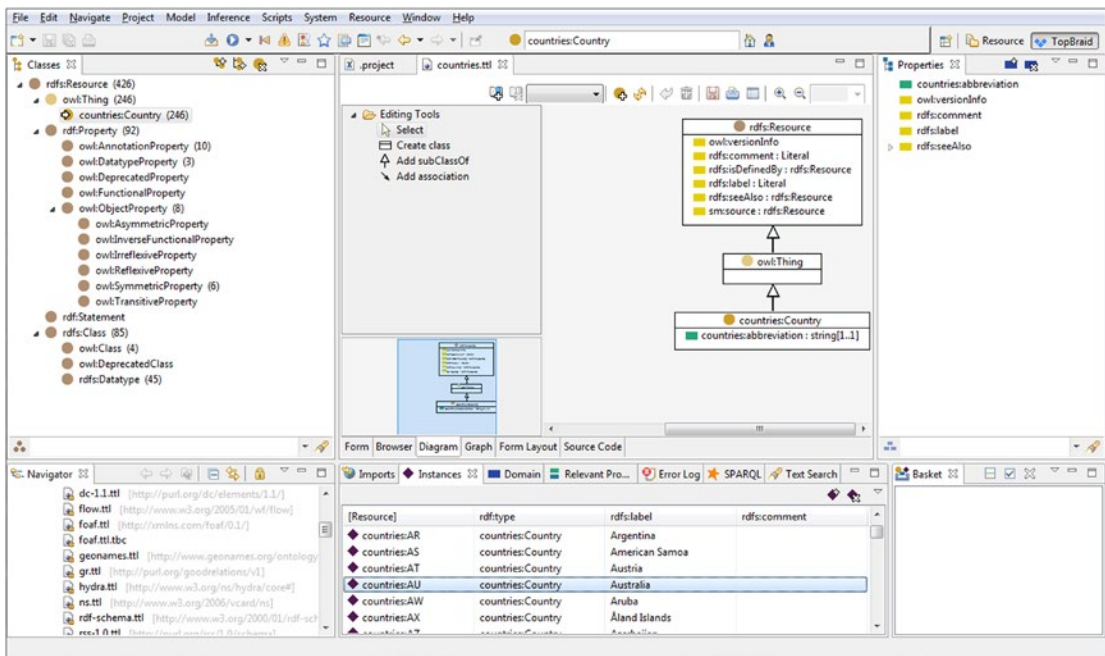
## SemanticWorks

Altova's *SemanticWorks* is a visual Semantic Web editor that features a graphical RDF and RDFS editor and a graphical OWL editor, supports OWL-Lite, OWL-Full, and OWL-DL dialects [10]. *SemanticWorks* provides syntax and format checking options and the evaluation of ontology semantics with direct links to errors. Context-sensitive entry helpers display the list of valid input options, depending on the serialization being used. *SemanticWorks* can generate code in RDF/XML and N-Triples and convert RDF/XML to N-Triples and vice versa. The program features a printing option for RDF and OWL diagrams. New class instances can be defined using intelligent shortcuts. The instances, properties, and classes are organized on tabs, and, similar to software engineering environments, properties and property values can also be manipulated through separate subwindows. The Overview subwindow is very useful when editing large, complex diagrams, when the currently displayed portion of the diagram is indicated as a red rectangle. You can switch between the diagram and the code view at any time.

## TopBraid Composer

TopQuadrant's *TopBraid Composer* is a graphical development tool for data modeling and semantic data processing. The free Standard Edition supports standards such as RDF, RDFS, OWL, and SPARQL, as well as visual editing and querying, and data conversions [11]. The commercial Maestro Edition provides a model-driven application development environment [12]. Composer is also an RDFizer, which can convert Excel spreadsheets into instances of an RDF schema.

TopBraid Composer can open ontologies serialized in RDF/XML or Turtle, import RDFa data sources, RSS or Atom news feeds, and e-mails into RDF. It can connect to SPARQL endpoints as well as RDBMS sources, import tab-delimited spreadsheet files and Excel spreadsheets, online RDF and OWL files, UML files, XML Schemas, and XML catalogs. Wizards guide you in creating new projects, such as faceted project resources, projects from CSV files, JavaScript projects, static web projects, as well as XML editing and validation. You can create markup files with RDFa and HTML5 Microdata annotations and develop semantic web applications and RDF/OWL file connections to Jena SDB databases, Jena TDB databases, Oracle databases, and Sesame 2 repositories. The Graphical User Interface features panels for classes, visual representation (diagrams and graphs) and source code, properties, file system navigation, imports, and “baskets” (see Figure 4-9).



**Figure 4-9.** Ontology editing with TopBraid Composer Maestro

On the Classes panel, you can navigate in your ontologies, represented as a tree structure, create and delete classes, create subclasses and siblings, group components by namespace, and search by name. The Properties panel features, among property manipulation, GoogleMaps integration too. On the Imports panel, the resources can be displayed, along with their `rdf:type`, `rdfs:label`, and `rdfs:comment` values (if provided), as well as rules, instances, errors, SPARQL queries, and text searches. On the Baskets panel, you can load contents from, and save contents to, a text file; add selected resources; add matching properties; add subclasses, subproperties, instances, individuals, and unreferences resources; and perform batch operations.

## Apache Stanbol

*Apache Stanbol* is a semantic data modeler and comprehensive ontology manager [13].

It includes a content-management system that supports Semantic Web services and web application functions such as tag extraction, text completion in search fields, and e-mail routing, based on extracted entities. The functionalities of the Stanbol components are available through a RESTful web service API. The RESTful services return results in RDF, JSON, and JSON-LD. Apache Stanbol can be run as a stand-alone application (packaged as a runnable JAR) or as a web application (packaged as .war) deployable in servlet containers such as Apache Tomcat. It is compatible with Apache frameworks such as Solr (for semantic search), Tika (for metadata extraction), and Jena (for storage).



Stanbol has a built-in RDFizer that processes traditional web contents sent in a POST request with the MIME type specified in the Content-type header and adds semantic information (“RDF enhancement”) to it, serialized in the format specified in the Accept header.

Stanbol also provides a reasoner component, which implements a common API and supports different reasoners and configurations through OWLApi and Jena-based abstract services, with implementations for Jena RDFS, OWL, OWLMini, and HermiT. The reasoner module can perform a consistency check, which returns HTTP Status 200 if data is consistent and 204 if not. The reasoner can also be used for classification, in other words, to materialize all inferred `rdf:type` statements. The semantic enrichment materializes all inferred statements.

The Apache Stanbol Ontology Manager supports multiple ontology networks by interconnecting seemingly unrelated knowledge represented in different ontologies, ontology libraries, a central ontology repository, as well as common ontology engineering tasks, such as reasoning and rule execution. Stanbol can also store and cache semantic information and make it searchable through its persistence services.

## Fluent Editor

*Fluent Editor* is an ontology editor, which can handle RDF, OWL, and SWRL files [14]. Fluent Editor uses a proprietary representation language and query language compatible with Semantic Web standards. The tool has been designed for managing complex ontologies. It features a reasoner window, a SPARQL window for queries, an XML preview window, a taxonomy tree view, and an annotation window. Fluent Editor has two types of plug-ins: a Protégé interoperability plug-in, which supports data export to and import from Protégé, and R language plug-ins that support the development of analytical models with R and rOntorion and plug-in development for Fluent Editor with the R language.

## Ontology Analysis Tools

There are software tools for ontology mapping and specific ontology engineering tasks not supported by general-purpose ontology editors such as semantic similarity estimation.

### ZOOMA

*ZOOMA* is an application for discovering optimal ontology mappings and automatic mapping of text values to ontology terms using mapping repositories [15]. *ZOOMA* can reuse mappings already asserted in the database, explore mapping best suitable for multiple mappings, derive better mappings by recording contextual information, and suggest new terms. The commonly observed values can be processed automatically.

ZOOMA finds all optimal mappings automatically where one text value maps to the same set of terms every time. When using mapping repositories, it can detect errors, in other words, it finds all the text value to ontology term mappings that are potentially incorrect. ZOOMA can also propose new mappings to terms based on the input values; however, selecting the best mapping requires human evaluation and assessment. ZOOMA can easily be used as a software library, as, for example, within an Apache Maven project.

## Semantic Measures Library

The *Semantic Measures Library (SML)* is a Java library for semantic measure analysis, such as estimating semantic similarity and relatedness by using ontologies to define the distance between terms or concepts [16]. SML functionalities can be accessed also through a set of command-line tools called *SML-Toolkit*. The library supports RDF and RDFS, OWL ontologies, WordNet (a lexical database), Medical Subject Headings (MeSH, a controlled vocabulary for life science publishing), the Gene Ontology, and so on.

## Reasoners

*Reasoners* derive new facts from existing ontologies and check the integrity of ontologies. The various software tools are different in terms of reasoning characteristics, practical usability, and performance, owing to the different algorithms implemented for Description Logic reasoning. Not all reasoners can evaluate all possible inferences, so their soundness and completeness vary. Some ontologies support rules for combining ontologies with rules. A common feature of reasoners is *ABOX reasoning*, the reasoning of individuals that covers instance checking, conjunctive query answering, and consistency checking. Advanced reasoners support the OWL API, a standard interface for application development with OWL reasoning. Another feature of advanced reasoners is the OWLLink support, leveraging an implementation-neutral protocol to interact with OWL 2 reasoners.

## HermiT

*HermiT* is one of the most popular OWL 2 reasoners that can be used to determine ontology consistency, identify relationships between classes, and perform further tasks [17]. HermiT uses its own algorithm, called the “*hypertableau*” calculus, to check the consistency of OWL ontologies and identify subsumption relationships between classes. HermiT can be used as a Protégé plug-in (see Figure 4-10), through the command line, or in Java applications. The latest Protégé versions come with a preinstalled HermiT plug-in. From the command line, you can perform classification, querying, and other common reasoning tasks. As for the Java applications, HermiT supports the OWLReasoner interface from the OWL API, providing access to OWL API objects, such as ontologies and class expressions.



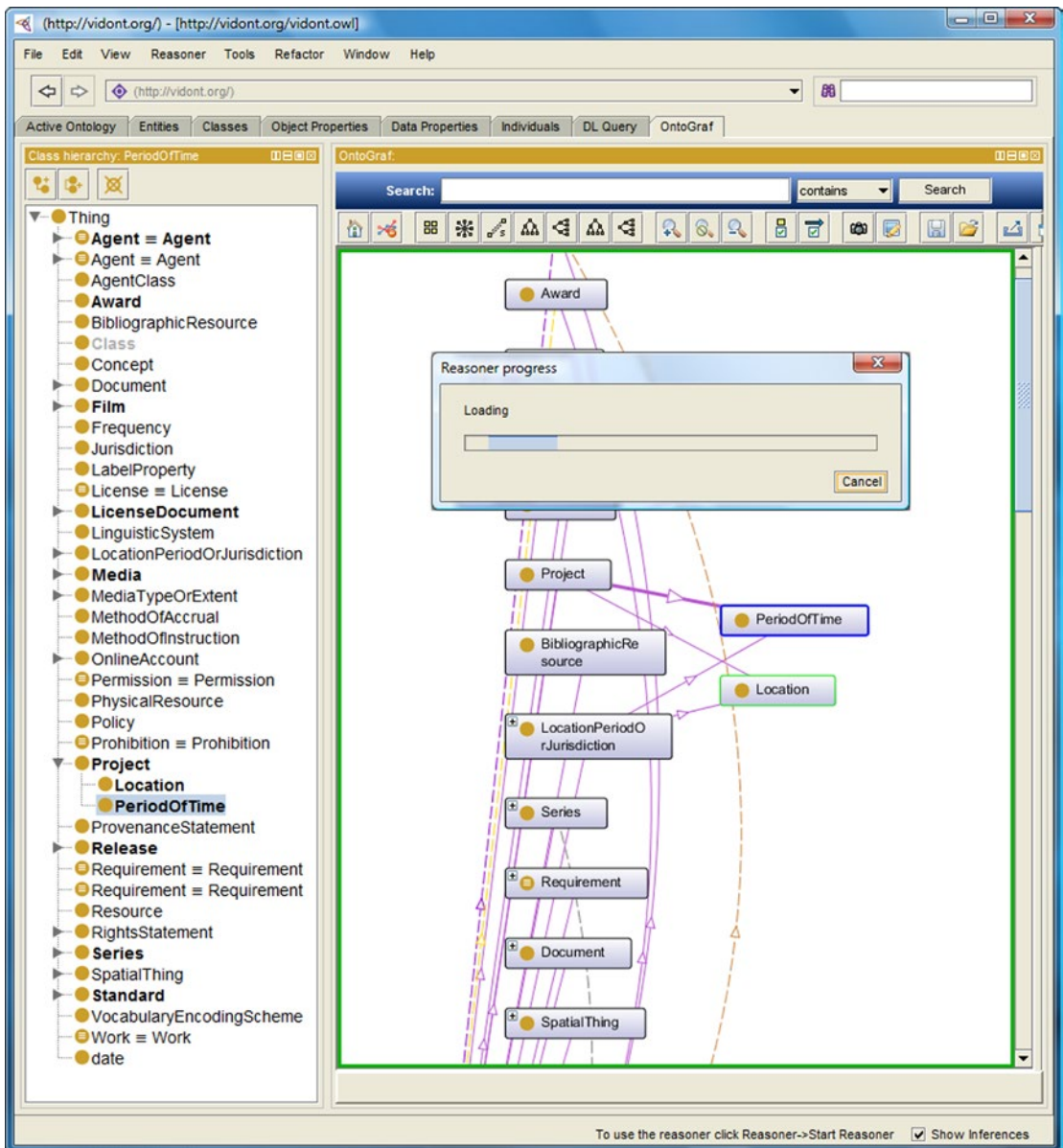


Figure 4-10. The HermiT reasoner running in Protégé

## Pellet

Clark & Parsia's *Pellet* is an OWL 2 DL reasoner, which can be used in Protégé, Jena, TopBraid Composer, or in Java programs through the OWL API interface [18]. It is based on the *tableau algorithm* to break down complex statements into smaller and simpler pieces to detect contradictions and supports expressive Description Logics. Pellet supports different incremental reasoning, including incremental consistency checking and incremental classification, where updates (additions or removals) can be processed and



applied to an ontology without having to perform all the reasoning steps from scratch. Pellet also supports reasoning with SWRL rules. It provides conjunctive query answering and supports SPARQL queries. Pellet reasons ontologies through Jena and the OWL API. Pellet also supports the explanation of bugs.

## FaCT++

*FaCT++* (Fast Classification of Terminologies) is a tableaux-based OWL 2 DL reasoner<sup>3</sup> [19]. It can be used as a description logic classifier and for modal logic satisfiability testing. It implements a sound and complete tableau algorithm for expressive description logics. *FaCT++* is available as a stand-alone tool, as a Protégé plug-in, and can be used in applications through the OWL API.

## RACER

*Racer* (Renamed ABox and Concept Expression Reasoner) is a server-side reasoner for building ontology-based applications, available through Java and Common Lisp APIs [20]. *Racer* provides not only standard reasoning mechanisms but also logical abduction. It implements a highly optimized tableau calculus for the Description Logic *SRIQ*<sup>(D)</sup>. *Racer* supports the consistency check of RDF data descriptions and OWL 2 ontologies and can open multiple ontologies simultaneously for ontology merging. It can find implicit subclass relationships induced by the axioms of an ontology and find synonyms for properties, classes, or instances. *Racer* can retrieve information from OWL/RDF documents via SPARQL queries and also support incremental queries. It supports *FaCT* optimization techniques and optimization for number restrictions and ABoxes.

# Application Development Frameworks

The most common programming tasks are collected in software libraries, so that you do not have to write frequently used code. In Semantic Web applications, for example, a common task is to convert an RDF file from one serialization to another, which can be easily performed by tools such as Apache Jena. Such software libraries can be used in a variety of environments, such as through the command line or as a plug-in of an Integrated Development Environment (IDE) such as Eclipse or NetBeans.

## Jena

Apache *Jena* is an open source Semantic Web and Linked Data application development framework, which supports the storage, retrieval, and analysis of structured data written in RDF [21].

The core RDF API of Jena has dedicated methods for extracting subjects, objects, and predicates of RDF statements such as `getSubject()`, which returns the `Resource`, `getObject()`, which returns the `RDFNode`, and `getPredicate()`, which returns the `Property` of the statement. Using the Jena RDF API, you can easily create and manipulate RDF graphs, which are called *models* in Jena and represented by the `Model` interface. For example, to describe a person using the RDF API, first define the URI or the subject and the string of the object (see Listing 4-1), then create an empty, memory-based `Model` using the `createDefaultModel()` method (see Listing 4-2).



### Listing 4-1. Constant Declaration in Jena

```
static String personWebsite = "http://www.lesliesikos.com";
static String personName = "Leslie Sikos";
```

<sup>3</sup>*FaCT++* has a partial support for OWL 2 key constraints and datatypes.

**Listing 4-2.** Creating a Memory-Based Model

```
Model model = ModelFactory.createDefaultModel();
```

The resource will be created using the `Model` (see Listing 4-3).

**Listing 4-3.** Creating a Resource

```
Resource lesliesikos = model.createResource(personWebsite);
```

Finally, add a property to the resource using `addProperty` (see Listing 4-4).

**Listing 4-4.** Adding Property to a Resource

```
lesliesikos.addProperty(FOAF.Name, personName);
```

To retrieve statements from an RDF graph (Jena Model), the `listStatements()` method can be used (see Listing 4-5).

**Listing 4-5.** Extracting RDF Triples

```
StmtIterator iter = model.listStatements();
```

If you need more details, you can list all the predicated, subjects, and objects from the RDF graph, as shown in Listing 4-6.

**Listing 4-6.** Listing All Triple Components Individually

```
while (iter.hasNext()) {
    Statement stmt      = iter.nextStatement();
    Resource  subject   = stmt.getSubject();
    Property  predicate = stmt.getPredicate();
    RDFNode   object    = stmt.getObject();

    System.out.print(subject.toString());
    System.out.print(" " + predicate.toString() + " ");
    if (object instanceof Resource) {
        System.out.print(object.toString());
    } else {
        System.out.print(" \"" + object.toString() + "\"");
    }

    System.out.println(" .");
}
```

Jena supports SPARQL queries, including SPARQL, over the JDBC driver framework. In fact, it can serve RDF data over HTTP, using *Fuseki*, a SPARQL server that provides REST-style SPARQL HTTP update, SPARQL querying, and SPARQL update [22]. The Jena rules engine, along with other inference algorithms, can derive consequences from RDF models. The *Inference API* provides reasoning to expand and check triplestore contents. You can not only use built-in OWL and RDFS reasoners but also configure your own inference rules. The Jena *Ontology API* can work with data models, RDFS, and OWL, including partial support for OWL 1.1 features. Jena has its own *high performance triplestore* component called TDB, which stores triples directly to disk and can be directly accessed from a Java Virtual Machine. SQL DB provides a persistent

triplestore for Jena, using relational databases, namely, an SQL database for RDF data storage and querying. Jena supports advanced text and spatial search. Jena can be integrated into Eclipse, the popular software development environment for Java developers.

## Sesame

*Sesame* is an open source framework for RDF data analysis and SPARQL querying [23]. The approach implemented to the Sesame framework is different from other semantic frameworks in a way that it features an extensible interface and that the storage engine is segregated from the query interface. *Alibaba*, a Sesame API, is used for mapping Java classes to ontologies and generating Java source files from ontologies, making it possible to directly exploit RSS, FOAF, and Dublin Core from Java. Sesame provides its RDF triplestore as a Java web application (.war), which can be easily deployed to application servers such as Apache Tomcat or Eclipse Jetty. It supports both memory-based (*MemoryStore*) and disk-based (*NativeStore*) storage. The RDF triplestore provides a SPARQL query endpoint. Sesame can be integrated to software development environments such as Eclipse and Apache Maven.

The *Repository API* provides methods for data file uploading, querying, extracting, and manipulation. One of its implementations, *SailRepository*, translates calls to a SAIL implementation of your choice, while another implementation, *HTTPRepository*, offers transparent client-server communication with a Sesame server over HTTP. The HTTP Server, the topmost component of Sesame, has Java servlets for accessing Sesame repositories over HTTP. Using the Repository API of Sesame, you can create a local repository directly from your application, with the capability to store, query, and modify RDF data (see Listing 4-7).

### Listing 4-7. Creating a Basic Local Repository in Sesame

```
import org.openrdf.repository.Repository;
import org.openrdf.repository.sail.SailRepository;
import org.openrdf.sail.memory.MemoryStore;
...
Repository repo = new SailRepository(new MemoryStore());
repo.initialize();
```

This repository will use the main memory for data storage, which is by far the fastest RDF repository type. However, the created repository is volatile, meaning that the content is lost when the object is garbage collected or when the program execution is finished. For persistent storage, you need to save the data to a file (see Listing 4-8).

### Listing 4-8. Creating a Local Repository with File Storage in Sesame

```
import org.openrdf.repository.Repository;
import org.openrdf.repository.sail.SailRepository;
import org.openrdf.sail.nativerdf.NativeStore;
...
File dataDir = new File("/path/to/datadir/");
Repository repo = new SailRepository(new NativeStore(dataDir));
repo.initialize();
```

To create a repository with RDF Schema inferencing, you have to create a Repository object by passing it a reference to the appropriate Sail object (see Listing 4-9).

**Listing 4-9.** Creating a Repository with RDF Schema Inferencing

```
import org.openrdf.repository.Repository;
import org.openrdf.repository.sail.SailRepository;
import org.openrdf.sail.memory.MemoryStore;
import org.openrdf.sail.inferencer.fc.ForwardChainingRDFSInferencer;
...
Repository repo = new SailRepository(
    new ForwardChainingRDFSInferencer(
        new MemoryStore()));
repo.initialize();
```

If you use a remote Sesame server rather than a local one, the remote connection has to be set up by initializing the `RemoteRepositoryManager` (see Listing 4-10).

**Listing 4-10.** Initializing a `RemoteRepositoryManager`

```
import org.openrdf.repository.manager.RemoteRepositoryManager;
...
String serverUrl = "http://localhost:8080/openrdf-sesame";
RemoteRepositoryManager manager = new RemoteRepositoryManager(serverUrl);
manager.initialize();
```

The *Storage And Inference Layer API (SAIL)* separates storage and inference. The SAIL API is primarily used by triplestore developers. The *RIO API*, which stands for “RDF I/O,” contains parsers and writers for RDF serialization. The parsers can transform RDF files to statements, while the writers can transform statements to RDF files. The RIO API can be used independently from all the other Sesame components.

The *RDF Model API* defines the representation of RDF building blocks such as statements, URIs, blank nodes, literals, graphs, and models. The RDF statements are represented by the `org.openrdf.model.Statement` interface, in which each statement has a subject, predicate, object, and (optionally) a context. Each of these items is an `org.openrdf.model.Value`, which covers `org.openrdf.model.Resource` and `org.openrdf.model.Literal`. Each resource represents an RDF value that is either a blank node (`org.openrdf.model.BNode`) or a URI (`org.openrdf.model.URI`). Literals represent RDF literal values such as strings, dates, and integer numbers. New triples and values can be created using `org.openrdf.model.ValueFactory` (see Listing 4-11).

**Listing 4-11.** Using a Default `ValueFactory` Implementation

```
ValueFactory factory = ValueFactoryImpl.getInstance();
```

Once you obtain your `ValueFactory`, you can create new URIs, literals, and triples (see Listing 4-12).

**Listing 4-12.** Adding URIs, Literals, and Triples to a `ValueFactory` Implementation

```
URI webstand = factory.createURI("http://yourbookdataset.com/webstand");
URI title = factory.createURI("http://yourbookdataset.com/title");
Literal webstandsTitle = factory.createLiteral("Web Standards");
Statement titleStatement = factory.createStatement(webstand, title, webstandsTitle);
```

The *Graph API* represents an RDF graph as a Java object. The `org.openrdf.model.Graph` class handles RDF graphs from the Java code. Graphs can be created in two ways: writing them programmatically by adding statements to them or created using a construct query. Empty graphs can be obtained by creating a `GraphImpl` object (see Listing 4-13).

**Listing 4-13.** Creating an Empty Graph

```
Graph myGraph = new org.openrdf.model.impl.GraphImpl();
```

Next, the RDF statement components (subject-predicate-object) have to be created using the ValueFactory object (see Listing 4-14). This prepares the graph to support triples and adds the WebDesignBook subject, the Title predicate, and the Web Standards object to the graph.

**Listing 4-14.** Adding Triple Support to a Graph

```
ValueFactory myFactory = myGraph.getValueFactory();
String namespace = "http://www.foo.com/bar#";

URI mySubject = myFactory.createURI(namespace, "WebDesignBook");
URI myPredicate = myFactory.createURI(namespace, "Title");
Literal myObject = myFactory.createLiteral("Web Standards");

myGraph.add(mySubject, myPredicate, myObject);
```

Another option is to use the URIs directly to add properties (see Listing 4-15).

**Listing 4-15.** Using URIs Directly to Add Triples to a Graph

```
URI bookClass = myFactory.createURI(namespace, "Book");
URI rdfType = myFactory.createURI(org.openrdf.vocabulary.RDF.TYPE);
mySubject.addProperty(rdfType, bookClass);
```

## Integrated Development Environments

Integrated Development Environments (IDEs) provide an interface for efficient Semantic Web application development, including a source editor with syntax highlighting for a variety of programming languages, such as Java and Python. IDEs have wizards and built-in applications to simplify software development, file handlers, and other tools to support deploying, running, and testing applications. IDEs consist of a runtime system, a workbench, and other features, such as a remote debugger or data modeler.

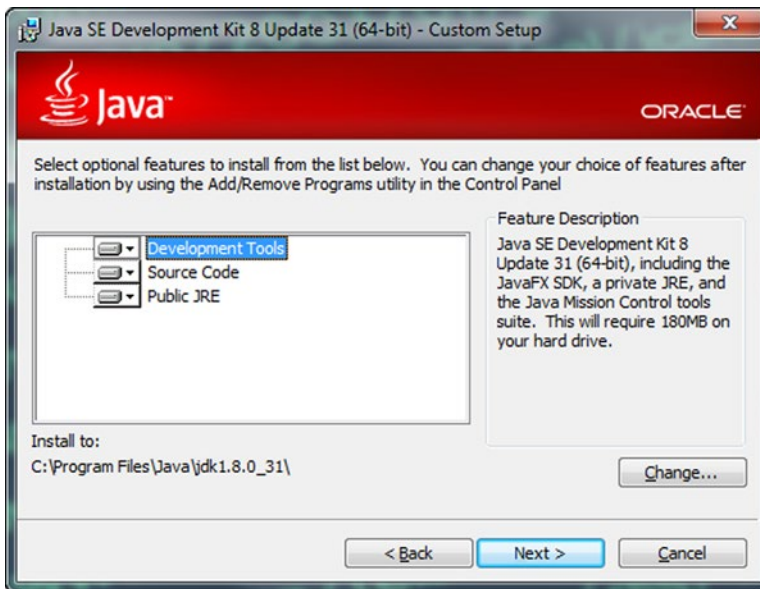
### Eclipse

*Eclipse* is one of the most popular IDEs for Java developers, providing essential tools, such as a Java IDE, a CVS client, a Git client, an XML Editor, and Apache Maven integration [24].



Eclipse is one of the popular IDEs to use Apache Jena and Sesame. The installation of Eclipse can be done as follows:

1. A prerequisite of Eclipse is the Java Development Kit (JDK). Download it from <http://www.oracle.com/technetwork/java/javase/downloads/> and install it (Figure 4-11).



**Figure 4-11.** Installing the Java Development Kit for Eclipse

---

■ **Caution** The Java Development Kit is different from the Java Runtime Environment (JRE), also known as the Java Virtual Machine (JVM), which is a secure computing environment for running Java programs on your computer.

---

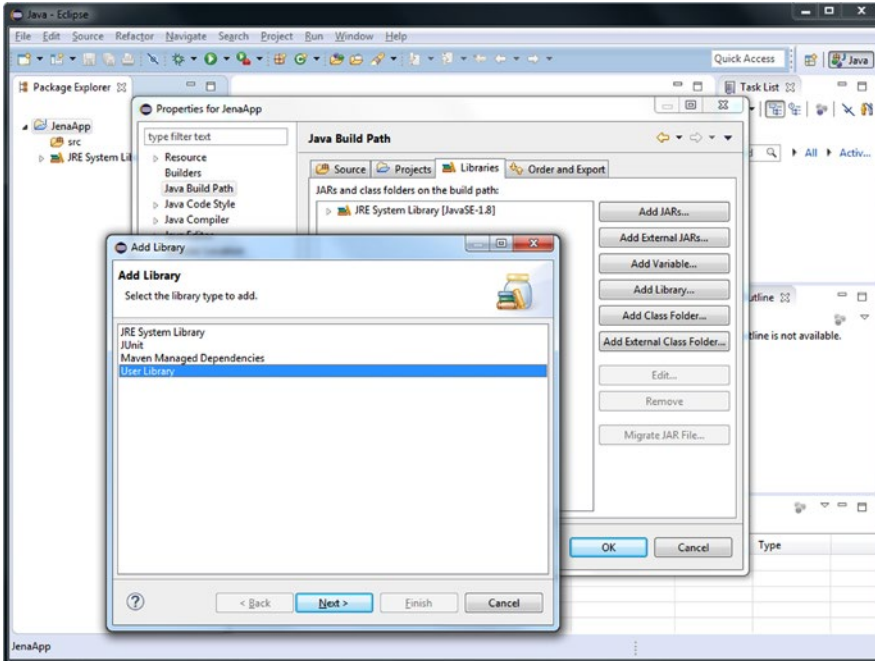
2. Visit <http://www.eclipse.org> and download the installer. Eclipse is available for Windows, Linux, and Mac OS X. The Windows binary is distributed as a ZIP archive, the Linux and the Apple installers as gzipped TAR archives.
3. Extract the installation files and execute `eclipse.exe`.
4. You have to specify a folder for Eclipse project files. If you want to use the same path every time you launch Eclipse, you can set the folder to the default Eclipse project folder.

## Set Up Apache Jena in Eclipse

Once you have Eclipse installed, you can set up Apache Jena.

1. Go to <http://jena.apache.org/download/>, select a download mirror, and download the binary distribution suitable for your platform (`.zip` or `.tar.gz`).
2. Extract the Jena files from the archive.
3. In Eclipse, select `File` ► `New` ► `Java Project`.
4. Right-click the name of the newly created project and select `Properties` (or select `File` ► `Properties`).

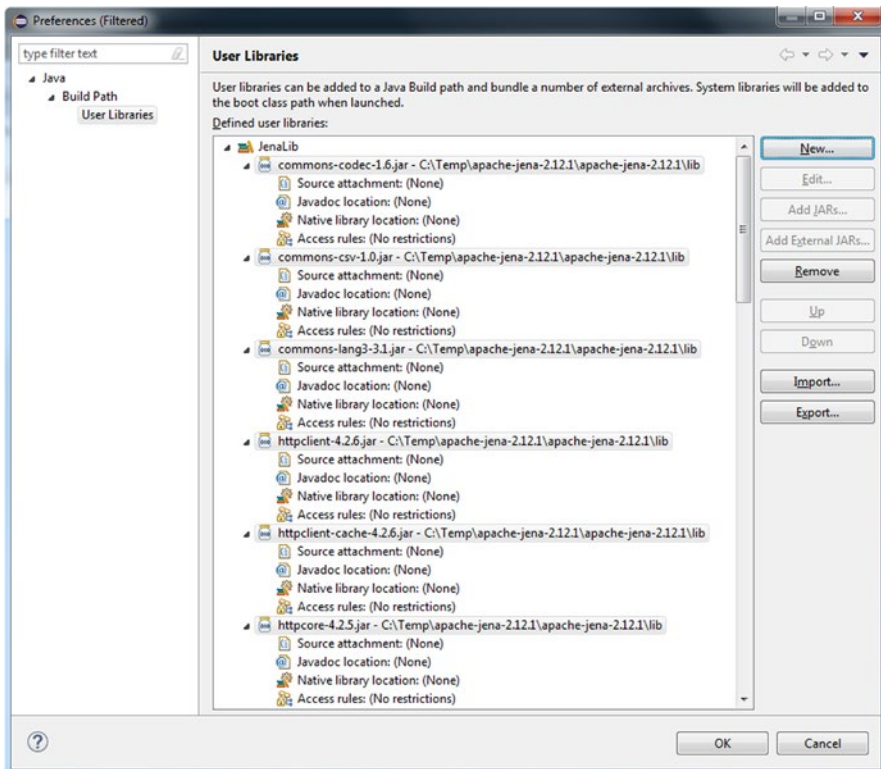
5. Select Java Build Path and click the Libraries tab.
6. Click Add Library... on the right.
7. Select User Library as the library type (see Figure 4-12).



**Figure 4-12.** Load the Apache Jena software library to Eclipse

8. Click the Next ► button on the bottom.
9. Click User Libraries... on the right.
10. Click the New... button.
11. Add a name to your library, such as JenaLib.
12. Click the Add external JARs... button on the right.
13. Browse to your Jena directory (apache-jena-versionNumber) and go to the lib subdirectory.
14. Select all the .jar files (for example, with Ctrl+A) and click Open (see Figure 4-13).
15. Click OK.
16. Click Finish.
17. Once you click OK, the Jena software library will be added to your Eclipse project.





**Figure 4-13.** Apache Jena to be added to the Eclipse project

To see the Jena library in action, let's create a Java program to convert your RDF/XML serialized FOAF file to Turtle!

1. In the Package Explorer, right-click src and select New ► Package and create a package.
2. Click the package name and select New ► File.
3. Specify a file name and click Finish.
4. Add the file content (type in directly or copy-paste it). If you don't have a FOAF file yet, create one manually in RDF/XML serialization or generate one using FOAF-a-matic at <http://www.ldodds.com/foaf/foaf-a-matic.html>. The asterisk (\*) in front of the file name on the file's tab indicates that the file has been changed. When you save the file with File ► Save or Ctrl+S, the character disappears. Save the file as, for example, foaf.rdf.

---

■ **Note** If you have characters not supported by Windows-1252, Eclipse offers you the option to save the file with UTF-8 encoding to avoid character loss.

---

5. Right-click the package and select New ► Class and add a name such as Main (creates Main.java).
6. Write the code to open the FOAF file and convert it to Turtle serialization using Apache Jena. Import the model (`com.hp.hpl.jena.rdf.model.Model`) and the File Manager of Jena (`com.hp.hpl.jena.util.FileManager`). Using the File Manager, load the model (`FileManager.get().loadModel()`) and write the RDF content out to the standard output (the console) in Turtle using `System.out` (see Listing 4-16).

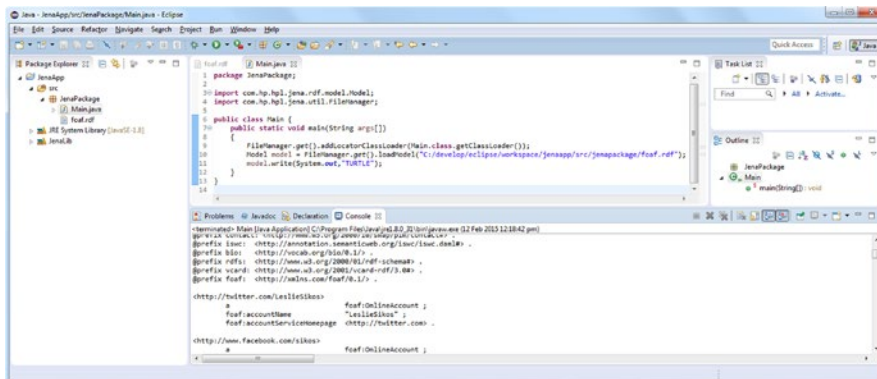
**Listing 4-16.** Loading and Converting an RDF File Using Jena

package JenaPackage;

```
import com.hp.hpl.jena.rdf.model.Model;
import com.hp.hpl.jena.util.FileManager;
```

```
public class Main {
    public static void main(String args[])
    {
        FileManager.get().addLocatorClassLoader(Main.class.getClassLoader());
        Model model = FileManager.get().loadModel("C:/develop/eclipse/workspace/
        jenaapp/src/jenapackage/foaf.rdf");
        model.write(System.out,"TURTLE");
    }
}
```

7. Run the program by clicking the Run button on the top toolbar (white triangle in green circle) or Run under the Run menu. The Console shows the output in Turtle (see Figure 4-14).



**Figure 4-14.** Using Apache Jena to convert RDF/XML to Turtle

## Set Up Sesame in Eclipse

Once you have Eclipse installed, you can add Sesame to your environment, similar to Jena.

1. Go to <http://sourceforge.net/projects/sesame/> and download the binary distribution.
2. Extract the Sesame files from the archive.
3. In Eclipse, select File ► New ► Java Project.
4. Right-click the name of the newly created project and select Properties (or select File ► Properties).
5. Select Java Build Path and click the Libraries tab.
6. Click Add Library... on the right.
7. Select User Library as the library type.
8. Click the Next > button on the bottom.
9. Click User Libraries... on the right.
10. Click the New... button.
11. Add a name to your library, such as JenaLib.
12. Click the Add external JARs... button on the right.
13. Browse to your Sesame directory (`openrdf-sesame-versionNumber`) and go to the `lib` subdirectory.
14. Select all the `.jar` files (for example, with `Ctrl+A`) and click Open (see Figure 4-15).

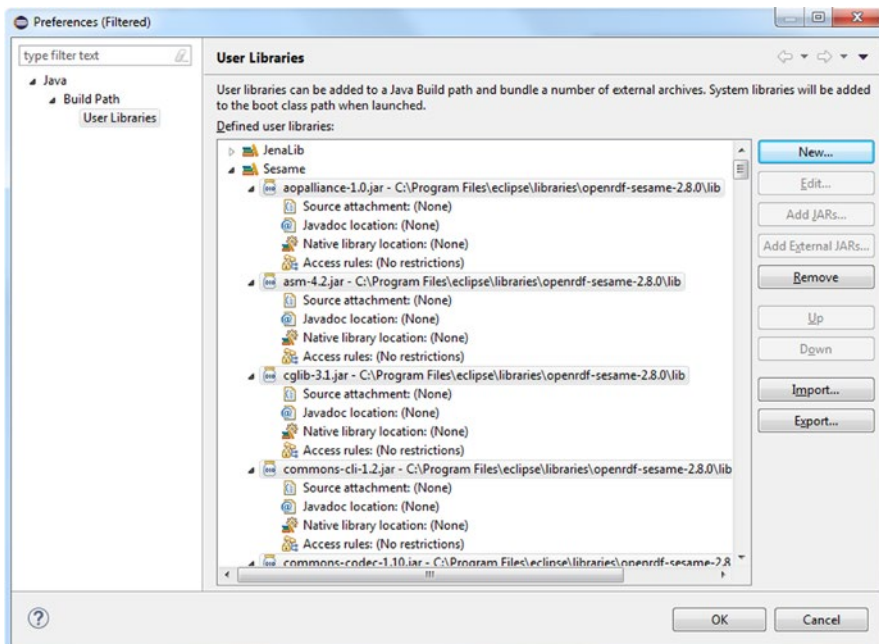


Figure 4-15. Adding Sesame to Eclipse

15. Click OK.
16. Click Finish.
17. Once you click OK, the Sesame software library will be added to your Eclipse project.

To see the Sesame library in action, let's create a Java program, to initialize a repository, and add data to and retrieve data from that repository!

1. Create a new Java class. To make it easier to write our code, on the New Java Class window, tick the checkbox `public static void main(String[] args)` under Which method stubs would you like to create?
2. To store RDF data, we first have to create a repository. While there are many different types of repositories, for our example, we need a simple local repository with fast in-memory store (see Listing 4-17).

**Listing 4-17.** Creating a Local Repository in Sesame

```
Repository rep = new SailRepository(new MemoryStore());
```

To use this code, however, we have to write some `import` statements manually (see Listing 4-18).

**Listing 4-18.** Import Packages from the Sesame Library

```
import org.openrdf.repository.Repository;
import org.openrdf.repository.sail.SailRepository;
import org.openrdf.sail.memory.MemoryStore;
```

Alternatively, you can force missing imports to be resolved automatically, using the `Ctrl+Shift+O` hot key.

3. Initialize the repository by calling the `rep.initialize()` method.
4. Add data to the repository. You can add triples directly from Java or load them from external files. In this example, we add some statements directly. To do so, we need a namespace to be used for creating new URIs and a `ValueFactory` for creating URI, `BNode`, and `Literal` objects (see Listing 4-19).

**Listing 4-19.** Adding Data to the Repository

```
String namespace = "http://example.com/";
ValueFactory f = rep.getValueFactory();
```

5. Create a new URI through an identifier for the person Leslie (see Listing 4-20).

**Listing 4-20.** Creating a URI

```
URI leslie = f.createURI(namespace, "leslie");
```

6. To add data to the repository, you have to open a `RepositoryConnection` (Listing 4-21).

**Listing 4-21.** Opening a Connection

```
RepositoryConnection conn = rep.getConnection();
```

7. To ensure that any connection is open only when needed, create a try-finally code block (see Listing 4-22). The try clause holds the tasks to be performed during a connection, while the finally clause is used to close the connection when it is not needed anymore or if something goes wrong.

**Listing 4-22.** A try-finally Block

```
try {
    }
finally {
    conn.close();
}
```

8. In the try clause, add triples to the repository (see Listing 4-23).

**Listing 4-23.** Adding RDF Statements to a Sesame Repository

```
conn.add(leslie, RDF.TYPE, FOAF.PERSON);
conn.add(leslie, RDFS.LABEL, f.createLiteral("Leslie",
XMLSchema.STRING));
```

The first triple describes Leslie as a Person, the second states Leslie's name as a string.

---

■ **Note** The frequently used namespaces (RDF, RDFS, FOAF, etc.) are predefined as constants in Sesame.

---

9. Retrieve the data from our repository using the `getStatements` method (see Listing 4-24), which has four arguments.

**Listing 4-24.** Data Retrieval from a Repository

```
RepositoryResult<Statement> statements = conn.getStatements(null,
null, null, true);
```

The first three arguments represent the subject, predicate, and object to be matched. In this case, we want to retrieve all triples. The first three arguments will be null. The last argument is a Boolean value for indicating whether those statements that are inferred by a reasoner should be included. In this example, we do not use any reasoners, so the fourth value won't have any effect on the output. Alternatively, one could use SPARQL queries as well, to extract data from the repository.

10. Convert the result to a Sesame Model (see Listing 4-25), which is a Java Collection.

**Listing 4-25.** Converting the Result to a Model

```
Model model = Iterations.addAll(statements, new LinkedHashModel());
```

11. To provide a neat output, we need some namespace abbreviations, so that the output won't include full URIs. Again, we can use the predefined constants for the RDE, RDFS, XMLSchema, and FOAF namespaces (see Listing 4-26).

**Listing 4-26.** Namespace Declaration

```
model.setNamespace("rdf", RDF.NAMESPACE);
model.setNamespace("rdfs", RDFS.NAMESPACE);
model.setNamespace("xsd", XMLSchema.NAMESPACE);
model.setNamespace("foaf", FOAF.NAMESPACE);
model.setNamespace("ex", namespace);
```

12. Display the output in Turtle on the Console, using the Sesame toolkit *Rio* ("RDF I/O") (see Listing 4-27).

**Listing 4-27.** Sending the Output to the Console

```
Rio.write(model, System.out, RDFFormat.TURTLE);
```

The final code should look like Listing 4-28.

**Listing 4-28.** A Complete Sesame Code Example

```
package sesamePackage;

import info.aduna.iteration.Iterations;

import org.openrdf.model.Statement;
import org.openrdf.model.URI;
import org.openrdf.model.Model;
import org.openrdf.model.ValueFactory;
import org.openrdf.model.impl.LinkedHashModel;
import org.openrdf.model.vocabulary.FOAF;
import org.openrdf.model.vocabulary.RDF;
import org.openrdf.model.vocabulary.RDFS;
import org.openrdf.model.vocabulary.XMLSchema;
import org.openrdf.repository.Repository;
import org.openrdf.repository.RepositoryConnection;
import org.openrdf.repository.RepositoryException;
import org.openrdf.repository.RepositoryResult;
import org.openrdf.repository.sail.SailRepository;
import org.openrdf.sail.memory.MemoryStore;
import org.openrdf.rio.RDFFormat;
import org.openrdf.rio.RDFHandlerException;
import org.openrdf.rio.Rio;
```

```

public class SesameApp {

    public static void main(String[] args) throws RepositoryException,
        RDFHandlerException {

        Repository rep = new SailRepository(new MemoryStore());
        rep.initialize();

        String namespace = "http://example.com/";
        ValueFactory f = rep.getValueFactory();

        URI leslie = f.createURI(namespace, "leslie");

        RepositoryConnection conn = rep.getConnection();
        try {
            conn.add(leslie, RDF.TYPE, FOAF.PERSON);
            conn.add(leslie, RDFS.LABEL, f.createLiteral("Leslie", XMLSchema.
                STRING));

            RepositoryResult<Statement> statements = conn.getStatements(null,
                null, null, true);

            Model model = Iterations.addAll(statements, new LinkedHashModel());
            model.setNamespace("rdf", RDF.NAMESPACE);
            model.setNamespace("rdfs", RDFS.NAMESPACE);
            model.setNamespace("xsd", XMLSchema.NAMESPACE);
            model.setNamespace("foaf", FOAF.NAMESPACE);
            model.setNamespace("ex", namespace);

            Rio.write(model, System.out, RDFFormat.TURTLE);
        }
        finally {
            conn.close();
        }
    }
}

```

Finally, you can run the application. The data stored in and retrieved from the repository is displayed on the Console (see Figure 4-16).



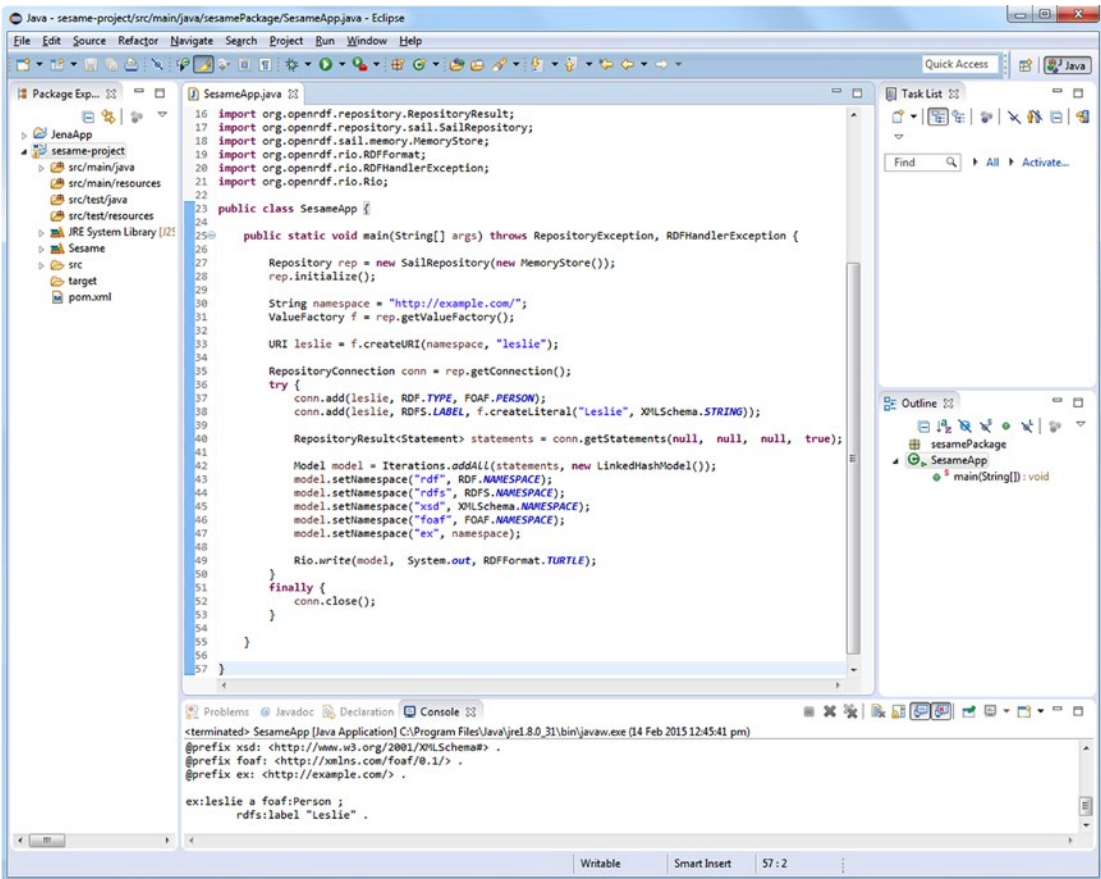


Figure 4-16. Using Sesame to store and retrieve RDF triples

## NetBeans

NetBeans is another popular Integrated Development Environment for Java [25].

NetBeans is powered by Apache Ant and supports Apache Maven, refactoring, version control, etc. All the functions of the IDE are provided through modules.



## Setup Apache Jena in NetBeans

Integrating Apache Jena in NetBeans is similar to the installation we discussed for Eclipse.

1. Go to <http://jena.apache.org/download/>, select a download mirror, and download the binary distribution suitable for your platform (.zip or .tar.gz).
2. Extract the Jena files from the archive.
3. In NetBeans, select File ► New Project ► JavaWeb.
4. Give a name to the project and select servers.
5. Select File ► Project Properties.
6. Select the Libraries category and select Add JAR/Folder.

7. Select the required files.
8. When the files are listed, verify and click OK.

To use Jena in your project, you have to import the required packages, as discussed in the previous sections. If you use Apache Maven integration in NetBeans, you can also start a Jena project as follows:

1. Select File ► New Project ► Maven ► Java Application.
2. Add a name to the project and additional information, such as location, then click Finish.
3. Once NetBeans has created a new Maven project and opened it, right-click Dependencies and choose Add Dependency....
4. Declare the Group ID, such as `org.apache.jena`, the Artifact ID, such as `jena-core`, and the version of your Jena integration.
5. Open the Dependencies directory and check the dependencies.

---

■ **Note** The download of declared dependencies can be forced by right-clicking Dependencies and choosing Download Declared Dependencies.

---

## CubicWeb

*CubicWeb* is an object-oriented Semantic Web application framework written in Python [26]. It supports RDF and OWL, features semiautomatic XHTML/XML/JSON/text generation, and a proprietary query language similar to SPARQL. CubicWeb supports SQL databases and LDAP directories. The rapid application development is powered by a library of reusable components called “cubes,” including a data model and views for common tasks. For example, the file cube contains the file entity type, gallery view functionality, and a file import utility. If you build a blog application, you create a new cube, such as *mycube*, and reuse the blog cube (see Figure 4-17).

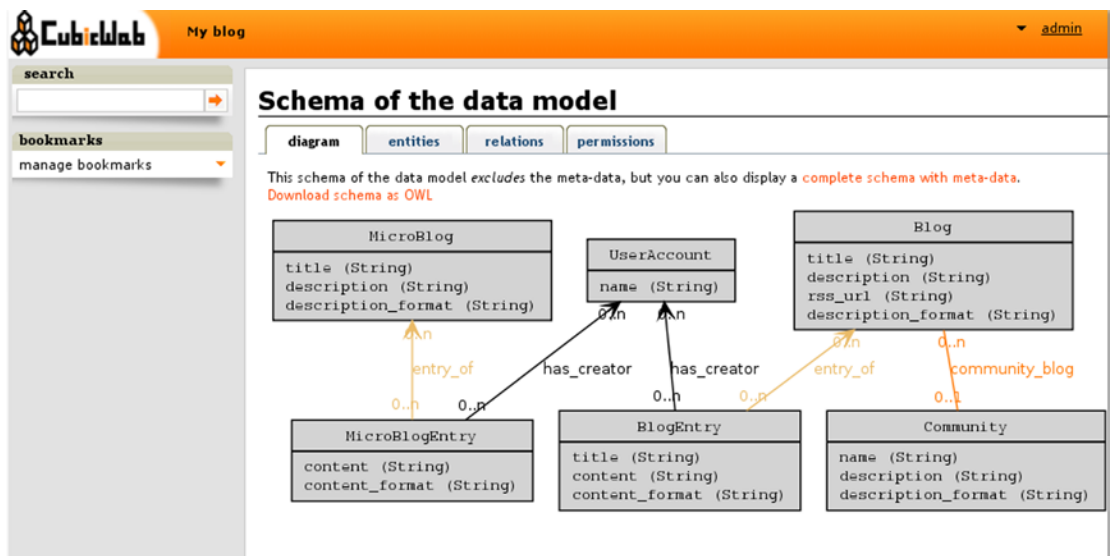


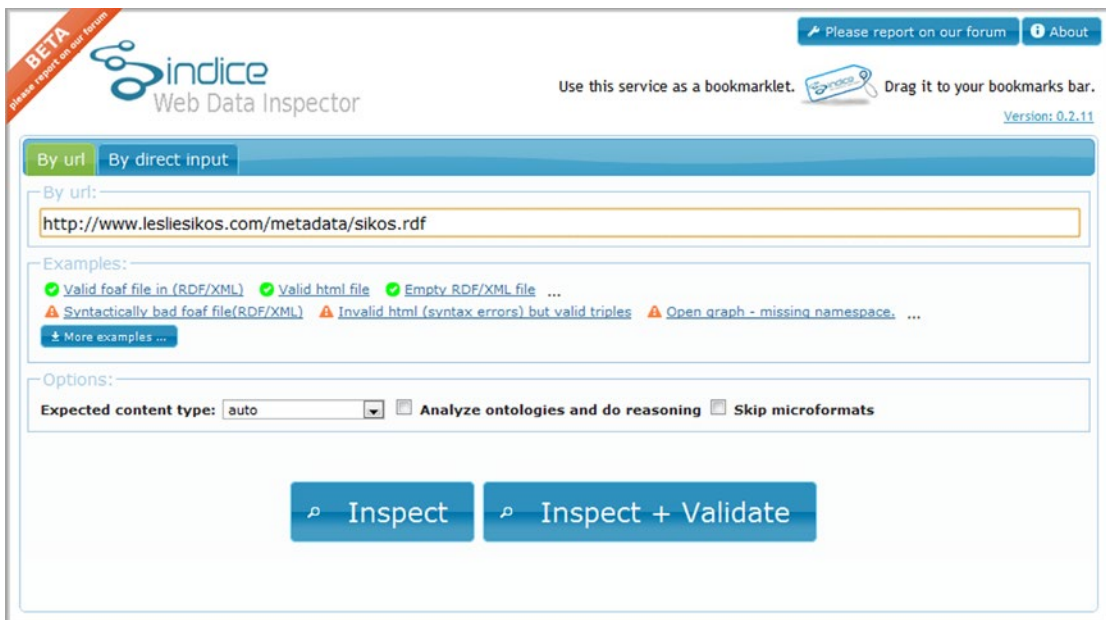
Figure 4-17. Visualization of the data model in CubicWeb

When you develop a new web application, you create a new cube, select building blocks from existing cubes, create class definitions in your cube’s schema, and create instances. The cubes have a standard folder structure to store the Python scripts, style sheets, JavaScript files, and translation files.

## Linked Data Software

### Sindice

*Sindice* is one of the most popular Linked Data platforms. Sindice collects, processes and integrates Linked Data from RDF, RDFa, Microformats, and HTML5 Microdata [27]. One of Sindice’s components is the *Sindice Web Data Inspector* at <http://inspector.sindice.com>, which is a comprehensive semantic data extractor tool. The tool can be used to extract RDF triples from markup, RDF/XML, Turtle, or N3 documents provided either by URI or by direct input. Sindice Web Data Inspector can be used for retrieving semantic data (Inspect button), combined semantic data extraction and validation (Inspect + Validate button), or ontology analysis and reasoning (see Figure 4-18).



**Figure 4-18.** Comprehensive options on the start screen of Sindice Web Data Inspector

As a result, the tool provides the full list of subject-predicate-object triples retrieved from the file. The output format can also be changed to N-triples or RDF/XML.

---

■ **Note** For usability reasons, Sindice Web Data Inspector displays a maximum of 1,000 triples only.

---

The “Sigma” option is a really good demonstration of machine-readable metadata. Software tools can extract structured data from properly written semantic documents and display them arbitrarily. This is the true essence of the Semantic Web !

A useful feature of Sindice Web Data Inspector is that a scalable graph can be generated from your semantic document. The graph not only presents the triples but also provides a quick summary of the ontologies and vocabularies used in the file.

The Sindice Web Data Inspector also has a validation feature with two different options. The first one, called “RDF syntax validation,” performs an RDF syntax validation according to the W3C specification. The second option is the “Pedantic validator,” which is a validation over the extracted triples. In case of a valid document, both validators give the result “Valid document.”

## Apache Marmotta

Apache *Marmotta* is a Linked Data server, SPARQL server, and Linked Data development environment [28]. Marmotta provides a Linked Data Platform (LDP) for human-readable and machine-readable read-write data access via HTTP content negotiation.

Marmotta features modules and libraries for LD application development. The modular server architecture makes it possible to implement the required functionalities only. For instance, if you don’t need reasoning for your project, you can exclude the reasoner module. Marmotta provides a collection of Linked Data libraries for common LD tasks such as access to LD resources and query Linked Data (through LDPPath, a simple LD query language). The triplestore is segregated from the server, so it can be used independently. The Apache Marmotta platform is implemented as a Java web application and deployed as a .war file. It is a service-oriented architecture using Contexts and Dependency Injection (CDI), a set of services of Java web application development. The *Marmotta Core*, a fundamental component of Apache Marmotta, provides Linked Data access, RDF import and export functionality, and an admin interface. Marmotta Core unites the service and dependency injection, the triplestore, the system configuration, and logging.



As a SPARQL server, Marmotta provides a public SPARQL 1.1 query and update endpoint and full support for SPARQL 1.1 through HTTP Web services. Marmotta features a fast, native SPARQL implementation in KiWi triplestore, a high-performance, highly scalable transactional triplestore back end for OpenRDF Sesame building on top of a relational database such as MySQL, PostgreSQL, or H2. To make SPARQL queries easier, Apache Marmotta provides Squebi, a lightweight user interface. Beyond KiWi, Marmotta’s default triplestore back end, you can also choose Sesame Native (based on Sesame Native RDF back end), BigData (based on the BigData clustered triplestore), or Titan (based on the Titan graph database). *Marmotta Reasoner*, an optional module, is a rule-based reasoner for the KiWi triplestore. It implements datalog-style rules over RDF triples. The *Marmotta Loader* is a command-line tool for loading RDF data in various formats to different triplestores. It supports RDF serializations and can also import directories, split-files, gzip and bzip2 compressed files, as well as Tar and Zip archives.

The Marmotta software libraries can be used not only as components of the Marmotta platform but also as stand-alone lightweight Java libraries. The Apache *Marmotta LDClient* library is a flexible and modular RDFizer suitable for Linked Data projects to retrieve remote Linked Data resources via different protocols and data providers [29]. The modules of Marmotta support RDF/XML, Turtle, N3, JSON-LD, RDFa, XML, HTML, and can process Freebase, Facebook, YouTube, Vimeo, and MediaWiki contents. The software library is extensible through Java’s *ServiceLoader* class, enabling custom wrappers for legacy data sources such as RDF, RDFa, Facebook, YouTube, and Wikipedia, as well as base classes for mapping other formats such as XML and JSON. *Marmotta LDCache*, another library, can access remote Linked Data resources as if they were local. It supports wrapping for legacy data sources such as Facebook Graph. LDCache features a local triplecache. Another optional library is *Marmotta LDPPath*, a query language less expressive than SPARQL but specifically designed for querying Linked Data in the cloud. LDPPath features a path-based navigation, which starts at the resource and follows the links.

---

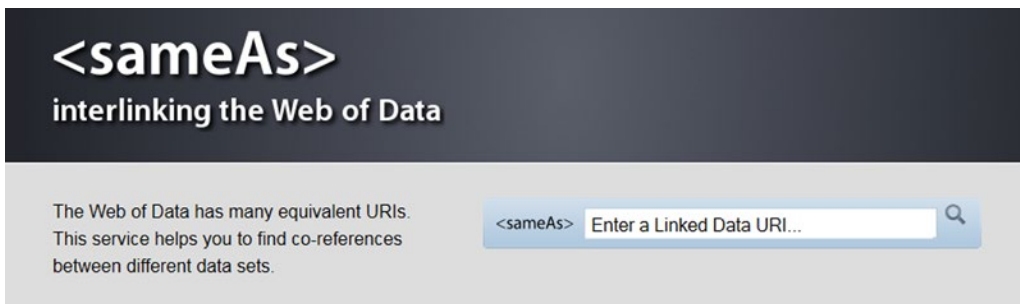
■ **Note** If you use SPARQL queries, LDPPath is recommended over LDCache.

---

LDPPath includes a large function library that can be integrated in your own applications. LDPPath can be used with LDCache and LDClient and supports back ends such as Jena and Sesame.

## sameAs.org

Because interlinking is fundamental in the Linked Open Data cloud, you will often define resources that describe the same object you express in RDF. For example, you refer to the machine-readable definition of your city of residence, pointing to its resource page on DBpedia, GeoNames, and Freebase. Because finding equivalent resource pages can be time-consuming, you might find the tool at [www.sameas.org](http://www.sameas.org), which finds equivalent resource pages across different datasets, useful (see Figure 4-19).



**Figure 4-19.** *sameAs* finds equivalent resource definitions from different LOD datasets

## Callimachus

*Callimachus*<sup>4</sup> is an integrated Linked Data application development environment for graph storage, visualization, RDFa templating, data processing with XSLT and XProc, SPARQL querying, and Linked Open Data publishing [30]. It is suitable for standardizing metadata and combining data from different systems and combining enterprise data with open data from the Web.

Callimachus extends the RDFa syntax by allowing variables as well as URIs to be used in attributes. Callimachus further extends the RDFa syntax by introducing expressions that allow values to be substituted within attribute values or within text nodes. Callimachus converts the attributes into graph patterns. Blank nodes and empty property contents are treated as wildcards. Graph patterns with wildcards or variables partners are optionally joined in the result.

## Neologism

*Neologism* is a free and open source vocabulary publishing platform [31]. It is distributed as a Drupal plug-in and supports RDF and RDFS and partially supports OWL. Neologism can import offline and online files written in RDF/XML, RDFa, Turtle, or OWL (see Figure 4-20). The form fields of Neologism feature client-side validation for correct input. Neologism displays relationships between terms in both directions. You can add arbitrary triples in Turtle to any vocabulary's RDF output.

---

<sup>4</sup>Named after Callimachus (310/305?–240 BC), the “Father of Bibliography,” who worked at the ancient Great Library of Alexandria.

**Import vocabulary**

**Namespace prefix:** \*

Examples: *foaf*, *dc*, *skos*. This will be used as both the ID and the namespace prefix for the imported vocabulary. It must be a prefix that is not yet in use.

**Namespace URI:** \*

Only classes and properties in this namespace will be imported! Must end in `~/"` or `~#`.

**Load vocabulary from the Web**

Use this to import a vocabulary from the Web. We will attempt to load an RDFS vocabulary or OWL ontology from the namespace URI.

**Load vocabulary from RDF file**

Use this to import a vocabulary from an RDF file on your computer. Select an RDF file (in RDF/XML format) that contains an RDFS vocabulary or OWL ontology.

**File upload:**

No file chosen  
Maximum file size is 2 MB.

**Figure 4-20.** *Neologism can import vocabulary and ontology files written in any of the mainstream RDF serializations*

## LODStats

*LODStats* is an extensible framework written in Python for high-performance dataset analytics [32]. It gathers statistical dataset characteristics such as class usage count, class hierarchy depth, property hierarchy depth, distinct entities, and so on. *LODStats* is so powerful that its developers integrated the framework with CKAN, the LOD cloud metadata registry to generate timely and comprehensive statistics about the LOD cloud.

## Semantic Web Browsers

Semantic Web browsers are browsing tools for exploring and visualizing RDF datasets enhanced with Linked Data such as machine-readable definitions from DBpedia or geospatial information from GeoData. Semantic Web browsers provide exploration, navigation, and interactivity features different from conventional web browsers. They display not only human-readable but also machine-readable annotations and extracted RDF triples. While conventional browsers use hyperlinks for navigating between documents, Semantic Web browsers provide mechanisms for forward and backward navigation with typed links. Semantic Web browsers support *facet-based (faceted) browsing* by processing the list of discrete filter attributes called *facets*, gradually refining the search on a collection of information and visualizing the result (such as generating a map from geospatial data). Semantic Web browsers also support *pivoting* (rotation), the dimensional orientation of data. For example, pivoting the initially aggregated *Book*, *Publisher*, and *Date* yields *Publisher*, *Date*, and *Book*. Semantic Web browsers can convert non-linked data to Linked Data and create links to related URIs. They provide text search or SPARQL queries, or both, and support the five-star data deployment scheme discussed in Chapter 3, for data consumption, generation, aggregation, augment, and reinterpretation.

## Tabulator

*Tabulator* is W3C's Semantic Web browser and editor available as a web application and a Firefox plug-in at <http://www.w3.org/2005/ajar/tab>. It can display Linked Data in various visualization formats. *Tabulator* contains an RDF store written in JavaScript. The tool has two modes: Exploration Mode and Query Mode.



In Exploration Mode, it displays a table of predicate-object pairs, which might also include nested properties. One of the exploration options is Outline Mode, with which the user can explore resources by opening the branches of the tree structure. The Outliner Mode addresses the limitations of the circle-and-arrow diagrams used by RDF visualizers, such as IsaViz, that are inefficient for large amounts of data with many nodes and many different properties. In Outliner Mode, the user can also perform graph-matching queries by selecting a number of fields and pressing the Find All button, when the Linked Data graph is searched for subgraphs matching the given fields. Instances are listed of a dedicated pane for each class. Tabulator can also show the network activity involved in retrieving the opened document, human-readable content, and RDF.

When used as an editor, Tabulator supports three editing options in Outline Mode: object modification, adding a new object with an existing predicate, and adding a new predicate-object pair to an existing subject. To modify a cell that contains a literal value, you click once (or press Enter) when the cell is highlighted, so that the field becomes editable. Once the editing is done, you just press Enter. If the object of the predicate-object pair is not a literal value but a URI identifier, you can select it by name or by drag-and-drop. Tabulator always tries to display a name rather than a URI whenever possible (for example, a textual description rather than `rdfs:label` or `dc:title`). When the predicate is not present, a new fact to the property or object table can be added by clicking the blue plus symbol displayed to the left, at the end of the table. When the new pair is added, you will be prompted with an auto-completion box for the predicate, while the object can be selected as usual.

When you perform a query for a subgraph pattern, a table is generated. Inserting a new row creates a new subgraph that matches the query. When a cell value is edited, a statement is removed and another inserted in the same document.

## Marbles

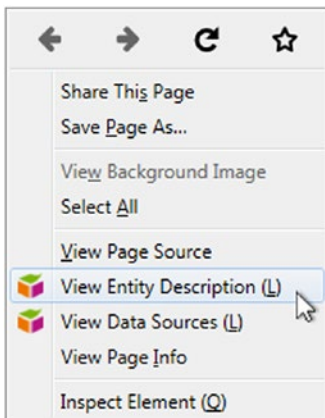
*Marbles* is a server-side application and linked data engine for semantic data retrieval and storage. As a Semantic Web browser, it displays colored “marbles” to indicate the relationship between data origin and data sources. Marbles can also be used as a SPARQL endpoint that supports SELECT, CONSTRUCT, and DESCRIBE queries. Once you download the .war file from <http://sourceforge.net/projects/marbles/files/>, you can place it into a J2EE web container such as Tomcat to install Marbles automatically. For manual installation, invoke the `ant install` and `remove` tasks on the source distribution, then invoke the `servlet` at the directory root. Among others, Marbles is implemented in DBpedia Mobile.

## OpenLink Data Explorer (ODE)

*OpenLink Data Explorer* (ODE, originally OpenLink RDF Browser) is a browser extension to exploit structured data. ODE adds two options to the standard View menu, both in the main menu and the context menu (see Figure 4-21).

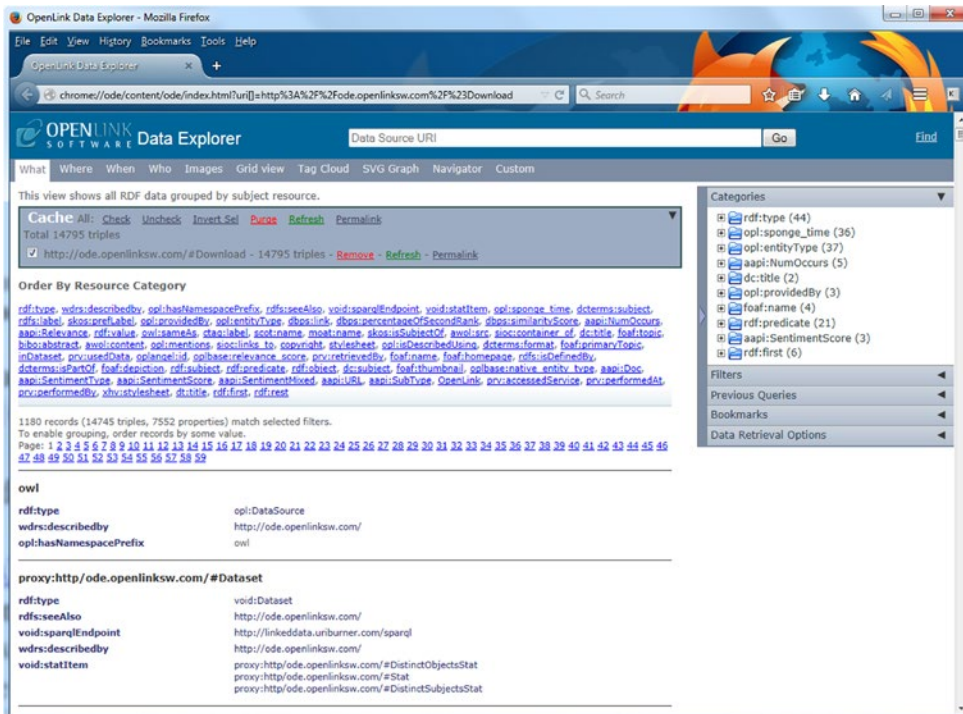
The Data Explorer is available for Internet Explorer, Firefox, Safari, Google Chrome, and Opera (<http://ode.openlinksw.com/#Download>). Let’s install the add-on, say, for Firefox!

1. Go to <http://s3.amazonaws.com/opldownload/ajax-tools/ode/1.1/firefox3.0/ode.xpi>.
2. Depending on your security settings, Firefox might prevent automatic installation. Click Allow to download the add-on.
3. The Software Installation pop-up asks for permission to proceed (“Install add-ons from authors whom you trust.”) Click Install Now.
4. Restart Firefox.



**Figure 4-21.** ODE options in the context menu

Once installed, the plug-in becomes available from the View menu as well as the context menu. The View Entity Description option gives a structured description of the current page. View Data Sources provides raw data display options for the structured data retrieved from the current page (see Figure 4-22).



**Figure 4-22.** Result screen of View Data Sources

The settings of the plug-in are available via Tools ► OpenLink Data Explorer ► Options. First, you can select the viewer. The default one is OpenLink Data Explorer, but you can also choose Zitgist Data Viewer, Marbles, DISCO, Tabulator, or a custom viewer. For Linked Data access, there is an RDFizer service,



a SPARQL Endpoint, and you can also define HTTP headers. The default host for RDFizer and the SPARQL endpoint is `linkeddata.uriburner.com`, which can be modified arbitrarily. The RDFizer is Virtuoso Sponger (<http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/VirtSponger>), a component of Virtuoso's SPARQL Processor and Proxy Web Service. Sponger supports RDFa, GRDDL, Amazon Web Services, eBay Web Services, Freebase Web Services, Facebook Web Services, Yahoo! Finance, XBRL Instance documents, Document Object Identifiers (DOI), RSS and Atom feeds, ID3 tags of MP3 music files, vCard, Microformats, Flickr, and Del.icio.us contents.

OpenLink Data Explorer handles RDF, Turtle, and Notation3 MIME data. The default viewer for MIME data is Virtuoso Describe, but you can also choose Virtuoso About or Virtuoso ODE with or without SSL.

## DBpedia Mobile

*DBpedia Mobile* is a location-aware DBpedia client application for mobile devices, providing a map view and a GPS-enabled launcher application [33]. Based on the GPS position of your mobile device, DBpedia Mobile displays a map that contains information about nearby locations extracted from the DBpedia dataset. Approximately 300,000 geographical locations are covered. DBpedia Mobile is powered by the rendering engine and SPARQL capabilities of the Marbles Linked Data Browser. Once the map is rendered, you can browse additional information about the location and go directly to DBpedia, GeoNames, Flickr, and other datasets.

## IsaViz

Being a visual authoring tool for RDF, *IsaViz* represents data as a circle-and-arrow diagram, which shows “things” related to each other (see Figure 4-23) [34]. This is useful when analyzing data structures. On an IsaViz diagram you can see clustering when a large number of “things” are related by the same properties.

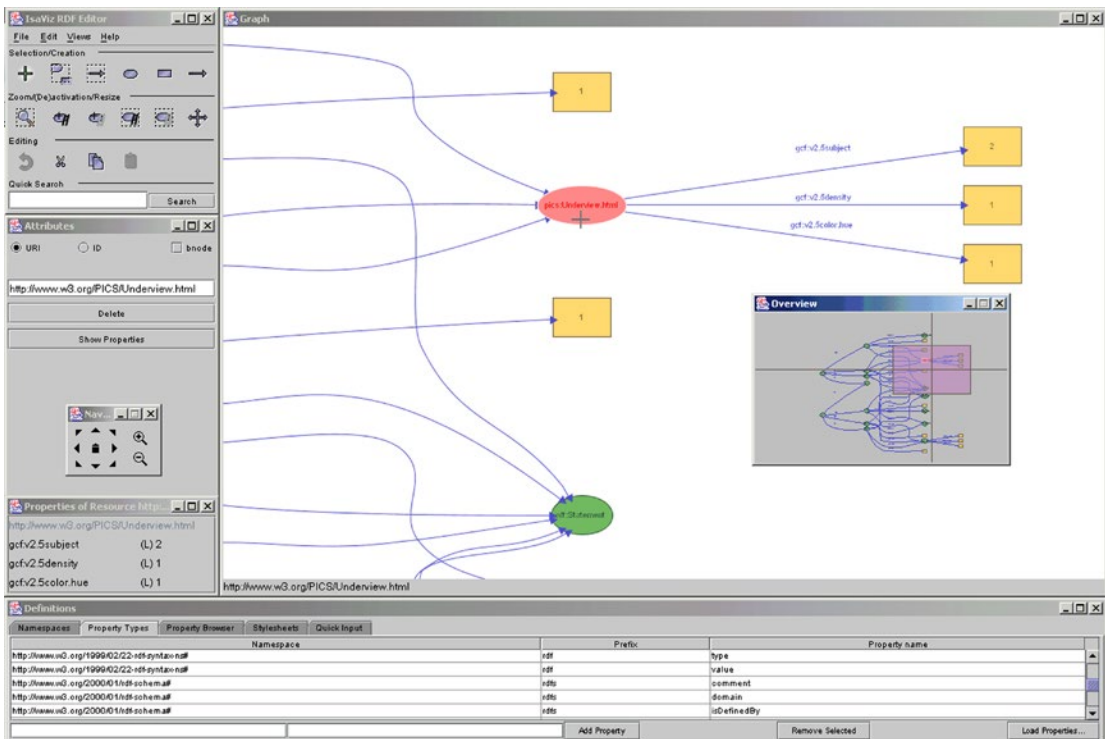
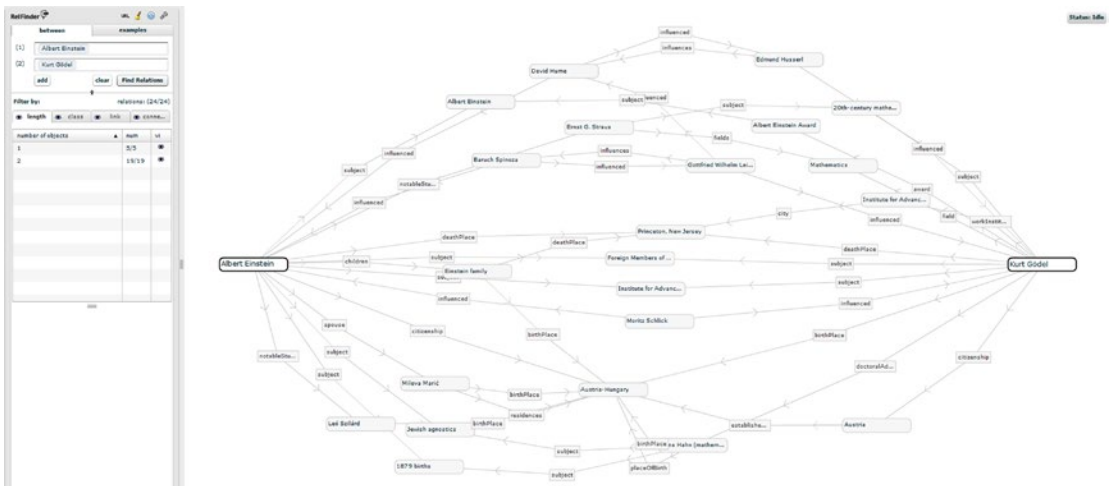


Figure 4-23. An RDF graph in IsaViz

## RelFinder

*RelFinder* (Relationship Finder) can be used to visualize, filter, and analyze large amounts of relationships between objects. It is suitable for knowledge representation and knowledge discovery. RelFinder provides standard SPARQL access to datasets. The online version is available at <http://www.visualdataweb.org/relfinder/relfinder.php>, which can generate a directed graph based on the selected objects and their relationships (see Figure 4-24).



**Figure 4-24.** Visualizing connections with RelFinder

## Summary

In this chapter, you became familiar with frequently used software tools that can generate, store, extract, and visualize RDF data. You learned how to generate RDFa annotation for your web site and test RDFa and Microdata annotations with Google Structured Data Testing Tool. You saw how to set up Integrated Development Environments to use software libraries for writing Semantic Web applications and examples for storing RDF data in, and retrieving RDF data from, repositories. You know the most popular Linked Data platforms to import, edit, and serialize datasets.

The next chapter will introduce you to Semantic Web service standards, including protocols, interfaces, and languages used by services such as location-aware applications and semantic e-commerce portals.

## References

1. Ho, D. et al. (2014) Notepad++. Don Ho. <http://notepad-plus-plus.org>. Accessed 31 March 2015.
2. Sessink, O. (2014) BlueFish. The Bluefish Project Team. <http://bluefish.openoffice.nl/>. Accessed 4 November 2014.
3. ActiveState Software (2014) Komodo. ActiveState Software. [www.activestate.com/komodo-ide](http://www.activestate.com/komodo-ide). Accessed 4 November 2014.

4. Bare Bones Software (2014) BBEdit. Bare Bones Software, Inc. [www.barebones.com/products/bbedit/](http://www.barebones.com/products/bbedit/). Accessed 4 November 2014.
5. Bare Bones Software (2014) TextWrangler. Bare Bones Software, Inc. [www.barebones.com/products/textwrangler/index.html](http://www.barebones.com/products/textwrangler/index.html). Accessed 4 November 2014
6. Lutus, P. (2014) Arachnophilia. [www.arachnoid.com/arachnophilia/](http://www.arachnoid.com/arachnophilia/). Accessed 4 November 2014.
7. GitHub (2015) DBpedia Spotlight. <https://github.com/dbpedia-spotlight/dbpedia-spotlight>. Accessed 31 March 2015.
8. The GATE project team (2015) GATE. <https://gate.ac.uk>. Accessed 31 March 2015.
9. The OpenRefine community (2015) OpenRefine. <http://openrefine.org>. Accessed 31 March 2015.
10. Altova (2012) Altova SemanticWorks 2012 User and Reference Manual. [www.altova.com/documents/SemanticWorks.pdf](http://www.altova.com/documents/SemanticWorks.pdf). Accessed 31 March 2015.
11. TopQuadrant (2015) TopBraid Composer Standard Edition. [www.topquadrant.com/tools/modeling-topbraid-composer-standard-edition/](http://www.topquadrant.com/tools/modeling-topbraid-composer-standard-edition/). Accessed 31 March 2015.
12. TopQuadrant (2015) TopBraid Composer Maestro Edition. [www.topquadrant.com/tools/ide-topbraid-composer-maestro-edition/](http://www.topquadrant.com/tools/ide-topbraid-composer-maestro-edition/). Accessed 31 March 2015.
13. The Apache Software Foundation (2015) Apache Stanbol. <http://stanbol.apache.org>. Accessed 31 March 2015.
14. Fluent Editor. [www.cognitum.eu/semantics/FluentEditor/](http://www.cognitum.eu/semantics/FluentEditor/). Accessed 15 April 2015.
15. The European Bioinformatics Institute (2015) ZOOMA. [www.ebi.ac.uk/fgpt/zooma/](http://www.ebi.ac.uk/fgpt/zooma/). Accessed 31 March 2015.
16. Harispe, S. (2014) Semantic Measures Library & ToolKit. [www.semantic-measures-library.org](http://www.semantic-measures-library.org). Accessed 29 March 2015.
17. Motik, B., Shearer, R., Glimm, B., Stoilos, G., Horrocks, I. (2013) Hermit OWL Reasoner. <http://hermit-reasoner.com>. Accessed 31 March 2015.
18. Clark & Parsia (2015) Pellet: OWL 2 Reasoner for Java. <http://clarkparsia.com/pellet/>. Accessed 31 March 2015.
19. Tsarkov, D., Horrocks, I. (2007) FaCT++. <http://owl.man.ac.uk/factplusplus/>. Accessed 31 March 2015.
20. University of Luebec (2015) Racer. [www.ifis.uni-luebeck.de/index.php?id=385](http://www.ifis.uni-luebeck.de/index.php?id=385). Accessed 31 March 2015.
21. The Apache Software Foundation (2015) Apache Jena. <http://jena.apache.org>. Accessed 31 March 2015.
22. The Apache Software Foundation (2015) Apache Jena Fuseki. <http://jena.apache.org/documentation/fuseki2/>. Accessed 31 March 2015.

23. Broekstra, J., Ansell, P., Visser, D., Leigh, J., Kampman, A., Schwarte, A. et al. (2015) Sesame. <http://rdf4j.org>. Accessed 31 March 2015.
24. The Eclipse Foundation (2015) Eclipse. [www.eclipse.org](http://www.eclipse.org). Accessed 31 March 2015.
25. Oracle Corporation (2015) NetBeans IDE. <https://netbeans.org>. Accessed 31 March 2015.
26. Logilab (2015) CubicWeb Semantic Web Framework. [www.cubicweb.org](http://www.cubicweb.org). Accessed 31 March 2015.
27. Digital Enterprise Research Institute (2015) Sindice—The Semantic Web index. <http://sindice.com>. Accessed 31 March 2015.
28. The Apache Software Foundation (2015) Apache Marmotta. <http://marmotta.apache.org>. Accessed 31 March 2015.
29. The Apache Software Foundation (2015) <http://marmotta.apache.org/ldclient/>. Accessed 31 March 2015.
30. 3 Round Stones (2015) Callimachus—Data-driven applications made easy. <http://callimachusproject.org>. Accessed 31 March 2015.
31. National University of Ireland (2011) Neologism—Easy Vocabulary Publishing. <http://neologism.deri.ie/>. Accessed 31 March 2015.
32. Auer, S., Ermilov, I., Lehmann, J., Martin, M. (2015) LODStats. <http://aksw.org/Projects/LODStats.html>. Accessed 1 April 2015.
33. Bizer, C. (2008) DBpedia Mobile. <http://wiki.dbpedia.org/DBpediaMobile>. Accessed 31 March 2015.
34. Pietriga, E. (2007) IsaViz: A Visual Authoring Tool for RDF. [www.w3.org/2001/11/IsaViz/](http://www.w3.org/2001/11/IsaViz/). Accessed 31 March 2015.