**8**

**LOGIC, RULES, AND INFERENCE**

**OVERVIEW**

The architecture of the Semantic Web is comprised of a series of layers that

form a hierarchy of content and logic. The ontology layer defines knowledge

about content, concepts, and relationships. Currently, the RDF Schema (RDFS)

is recognized as an ontology language that provides classes, properties, subsuperclasses,

range, and domain. However, RDFS has no localized range and

domain constraints, no cardinality constraints, no provision for negation, and

no transitive, inverse, or symmetrical properties. As a result, RDFS is unable to

provide sufficient expressive power for machine processing on the Semantic Web.

To expand the expressive capabilities of RDFS, three versions of the Web

Ontology Language (OWL) have been developed: OWL Full is the union of OWL

syntax and RDF, but it is undecidable, and therefore cannot provide complete

reasoning support. The OWL Descriptive Logic (DL) is a sublanguage of OWL

Full that has efficient reasoning support, but is not fully compatible with RDF.

Web Ontology Language Lite is an “easier-to-implement” subset of OWL DL.

Both RDF and OWL DL are specializations of predicate logic (also known as

first-order logic (FOL)) that are used for Web knowledge representation. They

provide a syntax that promotes their use on the Web in the form of tags, where

OWL DL and OWL Lite correspond roughly to a descriptive logic that is a subset

of predicate logic for which there exists adequate proof systems. Another subset

of predicate logic with efficient proof systems is the rule system Horn Logic.

*Thinking on the Web: Berners-Lee, G¨odel, and Turing*, by H. Peter Alesso and Craig F. Smith

Copyright 2006 John Wiley & Sons, Inc.

**143**

**144** LOGIC, RULES, AND INFERENCE

The choice of OWL version and complementary rule systems will dictate the

resulting computational complexity of the Semantic Web.

This chapter explains how logic and rules are used on the Semantic Web to

create inferences that manipulate and produce new knowledge. In addition, an

example of a simple RDF inference engine is provided.

**LOGIC AND INFERENCE**

Logic is the study of the principles of reasoning. As such, it constructs formal languages

for expressing knowledge, semantics, and automatic reasoners to deduce

(infer) conclusions.

Logic forms the foundation of Knowledge-Representation (KR), which has

been applied to Artificial Intelligence (AI) in general and the World Wide Web

in particular. Logic provides a high-level language for expressing knowledge

and has high expressive power. In addition, KR has a well-understood formal

semantics for assigning unambiguous meaning to logic statements.

Predicate (or first-order) logic, as a mathematical construct, offers a complete

proof system with consequences. Predicate logic is formulated as a set of axioms

and rules that can be used to derive a complete set of true statements (or proofs).

As a result, with predicate logic we can track proofs to reach their consequences

and also logically analyze hypothetical answers or statements of truth to determine

their validity. Proof systems can be used to automatically derive statements

syntactically from premises. Given a set of premises, such systems can analyze

the logical consequences that arise within the system.

Both RDF and OWL (DL and Lite) incorporate capabilities to express

predicate logic that provide a syntax that fits well with Web languages. They

offer a trade-off between expressive power and computational complexity (see

Chapter 2). Other subsets of predicate logic with efficient proof systems include

rules systems (e.g., Horn Logic or definite logic programs).

The Semantic Web language pyramid shown in Figure 2-2 identifies how the

ontology and logic layers fit together. An automatic reasoning system would

be formed on top of the ontology structure and it would make new inferences

through logic and proofs.

The top layer of the stack addresses issues of trust. This component of the

Semantic Web has not progressed far beyond a vision of allowing people to

ask questions of the trustworthiness of the information on the Web, in order to

provide an assurance of its quality.

**Inference Rules**

In logic, a rule is a scheme for constructing valid inferences. These schemes

establish syntactic relations between a set of formulas called premises and an

assertion called a conclusion. New true assertions can be reached from already

known ones.

LOGIC AND INFERENCE **145**

There are two forms of deductively valid argument: modus ponens (Latin

for “the affirming mode”) and modus tollens (the denying mode). Chapter 2

presented prominent examples of rules of inference in propositional logic starting

with the rules of modus ponens and modus tollens. For first-order predicate logic,

rules of inference are needed to deal with logical quantifiers.

Related proof systems are formed from a set of rules, which can be chained

together to form proofs, or derivations. If premises are left unsatisfied in the

derivation, then the derivation is a proof of a conditional statement: “*if* the

premises hold, *then* the conclusion holds.”

Inference rules may also be stated in this form: (*1*) some premises; (*2*) a

turnstile symbol , which means “infers,” “proves,” or “concludes”; and (*3*) a

conclusion. The turnstile symbolizes the executive power. The implication symbol

→ indicates *potential* inference and it is a logical operator.

For the Semantic Web, logic can be used by software agents to make decisions

and select a path of action. For example, a shopping agent may approve a discount

for a customer because of the rule:

RepeatCustomer*(*X*)*→ discount*(*25%*)*

where repeat customers are identified from the company database.

This involves rules of the form “IF (condition), THEN (conclusion).” With

only a finite number of comparisons, we are required to reach a conclusion. This

means that the logic will be tractable and the tools to execute it will be efficient

reasoning tools.

In addition, since the logic provides traceable steps in obtaining and backtracking

a conclusion, we can analyze the explanation for the premises and inference

rules used to reach the conclusion. Explanations are useful because they establish

validated proofs for the Semantic Web agents that provide credibility for their

results.

Axioms of a theory are assertions that are assumed to be true without proof. In

terms of semantics, axioms are valid assertions. Axioms are usually regarded as

starting points for applying rules of inference and generating a set of conclusions.

Rules of inference, or *transformation rules*, are rules that one can use to infer

a conclusion from a premise to create an argument. A set of rules can be used

to infer any valid conclusion if it is complete, while never inferring an invalid

conclusion, if it is sound.

Rules can be either conditional or biconditional. Conditional rules, or *rules of*

*inference*, are rules that one can use to infer the first type of statement from the

second, but where the second cannot be inferred from the first. With biconditional

rules, in contrast, both inference directions are valid.

**Conditional Transformation Rules**

We will use letters *p*, *q*, *r*, *s*, etc. as propositional variables.

**146** LOGIC, RULES, AND INFERENCE

An argument is Modus ponens if it has the following form (P1 refers to the

first premise; P2 to the second premise: C to the conclusion):

(P1) if *p* then *q*

(P2) *p*

(C) *q*

Example:

(P1) If Socrates is human then Socrates is mortal.

(P2) Socrates is human.

(C) Socrates is mortal.

Which can be represented as Modus ponens:

[*(p* → *q)* ∧ *p*] → [*q*]

An argument is Modus tollens if it has the following form:

(P1) if *p* then *q*

(P2) not-*q*

(C) not-*p*

Example:

(P1) If Socrates is human then Socrates is mortal.

(P2) Socrates is not mortal.

(C) Socrates is not human.

In both cases, the order of the premises is immaterial (e.g., in modus tollens

“not-*q*” could come first instead of “if *p* then *q*”).

Modus tollens

[*(p* → *q)*∧￢*q*] → [￢*p*]

An argument is a disjunctive syllogism if it has either of the following forms:

(P1) *p* or *q* (P1) *p* or *q*

(P2) not-*p* (P2) not-*q*

(C) *q* (C) *p*

The order of the premises is immaterial (e.g., “not-*q*” could come first instead

of “*p* or *q*”).

This argument form derives its name from the fact that its major premise is

a “disjunction,” that is, a proposition of the form “*p* or *q*.” The propositions *p*

and *q* are called the “disjuncts” of the disjunction “*p* or *q*.”

MONOTONIC AND NONMONOTONIC RULES **147**

In logic, the disjunction “*p* or *q*” is interpreted as the claim that not both *p*

and *q* are false; that is, that at least one of them is true. Thus a disjunction is held

to be true even when both its disjuncts are true. For example, the proposition

“either John ate breakfast this morning or he went running this morning” is true

even if John did both. Of course, the disjunction will also be true if John only

did one of the two. But if he did neither, then the disjunction is false.

Examples of disjunctive syllogism:

(P1) John ate breakfast or he went running.

(P2) John did not eat breakfast.

(C) John went running.

(P1) John ate breakfast or he went running.

(P2) John did not go running.

(C) John ate breakfast.

Conjunction introduction (or conjunction) is represented as

[*(p)* ∧ *(q)*] → [*p* ∧ *q*]

**Biconditional Transformation Rules**

Biconditional rules, or *rules of replacement*, are rules that one can use to infer

the first type of statement from the second, or vice versa.

Double negative elimination is represented as

[￢￢*p*] ↔ [*p*]

Tautology is represented as

[*p*] ↔ [*p* ∨ *p*]

**MONOTONIC AND NONMONOTONIC RULES**

If a conclusion remains valid after new information becomes available within

predicate logic, then we refer to this case as a monotonic rule. If, however, the

conclusion may become invalid with the introduction of new knowledge, then

the case is called a nonmonotonic rule.

The Semantic Web will express knowledge in a machine accessible way using

RDF and OWL, and then exchange rules across different applications using XMLbased

rule languages. A subset of predicate logic, Horn logic is the basis of

monotonic rules.

Nonmonotonic rules are useful where information is unavailable. These rules

can be overridden by contrary evidence presented by other rules. Priorities are

**148** LOGIC, RULES, AND INFERENCE

helpful to resolve some conflicts between nonmonotonic rules. The XML-based

languages can be used to represent rules.

**DESCRIPTIVE LOGIC**

Descriptive logic is a family of logic based on knowledge-representation formalisms

that is a descendant of semantic networks. It can describe the domain

in terms of concepts (classes), roles (properties, relationships), and individuals.

Descriptive logic is distinguished by being a formal semantic that has decidable

fragments of FOL and has provisions of inference services. Descriptive logics

allow specifying a terminological hierarchy using a restricted set of first-order

formulas. They usually have nice computational properties (often decidable and

tractable), but the inference services are restricted.

**Inference and Classes**

We can make inferences about relationships between classes, in particular subsumption

between classes. Recall that A subsumes B when it is the case that any

instance of B must necessarily be an instance of A.

**Inference and Individuals**

We can make inferences about the individuals, in particular inferring that particular

individuals must be instances of particular classes. This can be because

of subsumption relationships between classes, or because of the relationships

between individuals.

The Unique Name Assumption (UNA) says that any two individuals with

different names are different individuals. Many DL reasoners assume UNA, but

OWL semantics does not make use of the UNA. Instead there are mechanisms

in the language (owl:differentFrom and owl:AllDifferent) that allow us to assert

that individuals are different.

**Closed and Open Worlds**

Reasoning in DLs is monotonic. This means that if we know that *x* is an instance

of A, then adding more information to the model cannot cause this to become

false. We cannot assume that if we do not know something, then it is false. This

is due to the Open World Assumption (OWA).

**Simple Common Logic**

Computer-understandable ontologies are represented in logical languages, such

as the W3C OWL and the draft ISO standard, SCL (Simple Common Logic).

However, logical languages are only a means to express content. It is the information

being imparted in the statements that drives how the individual words

INFERENCE ENGINES **149**

are selected and sequenced into sentences. It is not the language (or logic) that

makes the difference, but how it is used. Ontology is one way to use language

and logic more effectively.

Simple Common Logic (SCL) is a proposal for a unified semantic framework

for expressing full first-order logical (FFOL) content for transmission on the

Web. Simple Common Logic was recently submitted for ISO standardization as

Common Logic, and has been incorporated into the OMG Ontology Definition

Metamodel (ODM) standard. The SCL extends conventional first-order notations

in various ways and is the candidate formalism for expressing content that is

currently represented in both description logics and rule languages.

**INFERENCE ENGINES**

An expert system has three levels of organization: a working memory, an inference

engine, and a knowledge base. The inference engine is the control of the

execution of reasoning rules. This means that it can be used to deduce new

knowledge from existing information.

The inference engine is the core of an expert system and acts as the generic

control mechanism that applies the axiomatic knowledge from the knowledge

base to the task-specific data to reach some conclusion.

Two techniques for drawing inferences are general logic-based inference

engines and specialized algorithms.

Many realistic Web applications will operate agent-to-agent without human

intervention to spot glitches in reasoning. Therefore developers will need to have

complete confidence in reasoner otherwise they will cease to trust the results.

Doubting unexpected results makes a reasoner useless.

**How the Inference Engine Works**

In simple rule-based systems, there are two kinds of inference, forward and

backward chaining.

**Forward Chaining**

In forward chaining, the data is put into working memory. This triggers rules

whose conditions match the new data. These rules then perform their actions.

The actions may add new data to memory, thus triggering more rules, and so on.

This is also called data-directed inference, because inference is triggered by the

arrival of new data in working memory.

Consider iterating continuously though the following set of rules until you

reach a conclusion:

Rule 1: IF A and C THEN F

Rule 2: IF A and E THEN G

Rule 3: IF B THEN E

Rule 4: IF G THEN D

**150** LOGIC, RULES, AND INFERENCE

To prove that D is true, given that A and B are true, we start with Rule 1 and

go on down the list until a rule that “fires” is found. In this case, Rule 3 is the

only one that fires in the first iteration. At the end of the first iteration, it can

be concluded that A, B, and E are true. This information is used in the second

iteration.

In the second iteration, Rule 2 fires adding the information that G is true. This

extra information causes Rule 4 to fire, proving that D is true.

This is the method of forward chaining, where one proceeds from a given

situation toward a desired goal, adding new assertions along the way. This strategy

is appropriate in situations where data are expensive to collect and few are

available.

**Backward Chaining**

In backward chaining the system needs to know the value of a piece of data.

It searches for rules whose conclusions mention this data. Before it can use the

rules, it must test their conditions. This may entail discovering the value of more

pieces of data, and so on. This is also called goal-directed inference, or hypothesis

driven, because inferences are not performed until the system is made to prove

a particular goal.

In backward chaining, we start with the desired goal and then attempt to find

evidence for proving the goal. Using the forward chaining example, the strategy

to prove that D is true would be the following.

First, find the rule that proves D. This is Rule 4. The subgoal is then to prove

that G is true. Rule 2 meets the subgoal, and as it is already known that A is

true, therefore the next subgoal is to show that E is true. Rule 3 provides the

next subgoal of proving that B is true. But the fact that B is true is one of the

given assertions. Therefore, E is true, which implies that G is true, which in turn

implies that D is true.

Backward chaining is useful in situations where the amount of data is large

and where a specific characteristic of the system is of interest. Typical situations

include medical diagnosis or fault finding in electrical equipment.

Some expert systems use more complex methods, for example, mixtures of forward

and backward chaining. Some have probability factors attached to rules. Yet

others store their rules in frames, and trigger them when an object is recognized

as matching that frame.

**Tree Searches**

A knowledge base can be represented as a branching network or tree. There is

a large number of tree searching algorithms available in the existing literature.

However, the two basic approaches are depth-first search and breadth-first search.

The depth-first search algorithm begins at a node that represents either the

given data (forward chaining) or the desired goal (backward chaining). It then

INFERENCE ENGINES **151**

checks to see if the left-most (or first) node beneath the initial node (call this

node A) is a terminal node (i.e., it is proven or a goal). If not, it establishes

node A on a list of subgoals outstanding. It then starts with node A and looks

at the first node below it, and so on. If there are no more lower level nodes,

and a terminal node has not been reached, it starts from the last node on the

outstanding list and takes the next route of descent to the right.

Breadth-first search starts by expanding all the nodes one level below the

first node. Then it systematically expands each of these nodes until a solution is

reached or else the tree is completely expanded. This process finds the shortest

path from the initial assertion to a solution. However, such a search in large

solution spaces can lead to huge computational costs due to an explosion in the

number of nodes at a low level in the tree.

There are other methods of making inferences that use a combination of two

or more of the above techniques. Depending on the number of given facts and

the number of plausible inferences, some of these methods may be better than

others in terms of time, memory, and cost of the solution path (see Chapter 12

for Semantic Search Technology).

**Full First-Order Logic Inference Engines**

Using full first-order logic for specifying axioms requires a full-fledged automated

theorem prover. First-order logic is semidecidable and inferencing is computationally

intractable for large amounts of data and axioms.

This means that in an environment such as the Web, these programs would

not scale up for handling huge amounts of knowledge. Besides, full first theorem

proving would mean maintaining consistency throughout the Web, which is

impossible.

The approach taken by CYCORPs CYC (see http://www.cyc.com/products.

html) is different. Their approach consists of roughly 1 MB of axioms using the

first-order framework. The CYC organizes its axioms in contexts and maintains

consistency just for one context, and it limits deductions to a few steps. Compared

to future Web architecture, CYC is still small.

An interactive theorem prover is not suitable for automated agents since they

rely on user interaction. However, they may be useful to construct proofs, which

can be validated by automated agents.

**Closed World Machine**

The Closed World Machine (CWM) (www.w3.org/2000/10/swap/doc/cwm.html)

inference engine written in Python by Tim Berners-Lee and Dan Connolly is a

popular Semantic Web program. It is a general-purpose data processor for the

Semantic Web and is a forward-chaining reasoner that can be used for querying,

checking, transforming, and filtering information. Its core language is RDF,

extended to include rules.

**152** LOGIC, RULES, AND INFERENCE

**RDF INFERENCE ENGINE**

This section presents the elements of a simple RDF inference engine. RDF is a

system meant for stating meta-information through triples composed of a subject,

a property, and an object. The subject and object can be either a designation like

a URL or a set of another triple. Triples form a simple directed graph.

Figure 8-1 shows a simple RDF example. The first triple says that Smith owns

a computer and the second says that there is a computer made by Apple. The

third drawing, however, is composed of two triples, and it says that Smith owns

a computer made by Apple.

Suppose these triples were placed in a database. Now we can conduct a query

as in Figure 8-2.

In the first query, the question is who owns a computer? The answer is “Smith.”

In the second query, the question is What make of computer are defined in the

Smith

Smith

computer

computer

computer Apple

Apple

owns

owns

is manufacture by

is manufacture by

**Figure 8-1.** RDF statements.

? who

? who

computer

computer

computer ? what

? what

owns

owns

is manufacture by

is manufacture by

**Figure 8-2.** RDF queries.

RDF INFERENCE ENGINE **153**

database? The third query, however asks who owns a computer and what is the

make of that computer?

The query is a graph containing variables that can be matched with the graph

in Figure 8-1. Should the graph in the database be more extended, it would have

to be matched with a subgraph. So, generally for executing an RDF query what

has to be done is called “subgraph matching.”

Following the data model for RDF the two queries are in fact equal because

a sequence of statements is implicitly a conjunction. Figure 8-3 illustrates this.

Let us make a rule: If X owns a computer, then X must buy software. How

do we represent such a rule? Figure 8-3 gives the graph representation of a rule.

The nodes of the rule form a triple set. Here there is one antecedent, but

there could be more. There is only one consequent. (Rules with more than one

consequent can be reduced to rules with one consequent.) Figure 8-4 gives a

query that will match with the consequent of the rule.

The desired answer is John must buy software. The query of Figure 8-4 is

matched with the consequent of the rule. Now an action has to be taken: The

antecedents of the rule have to be added to the database with the variables

replaced with the necessary values (substitution). Then the query has to be continued

with the antecedent of the rule.

The question now is Who owns a computer? This is equal to a query described

earlier. A rule subgraph is treated differently from nonrule subgraphs.

X computer

owns

Implies

X software

Must\_buy

Consequent

Antecedent

**Figure 8-3.** Graph representation of a rule.

? who Software

Must\_buy

**Figure 8-4.** Query that matches with a rule.

**154** LOGIC, RULES, AND INFERENCE

A triple can be modeled as a predicate: triple(subject, property, object). A set

of triples equals a list of triples and a connected graph is decomposed into a set

of triples. For our example this gives

Triple(John, owns, computer).

Triple(computer, make, Apple).

This sequence is equivalent to: [Triple(John, owns, computer). Triple(computer,

make, Apple).]

From Figure 8-2 the triples are

Triple(?who, owns, computer).

Triple(computer, make, ?what).

This sequence is equivalent to: [Triple(?who, owns, computer). Triple(computer,

make, ?what).]

From Figure 8-3 the triple is Triple([Triple(X, owns, computer)], implies,

[Triple(X, must buy, software)]).

From Figure 8-4 the triple is Triple(?who, must buy, software).

A unification algorithm for RDF can handle subgraph matching and embedded

rules by the term “subgraph matching with rules.” The unification algorithm

divides the sequence of RDF statements into sets where each set constitutes a

connected subgraph. This is called a tripleset that is done for the database and

for the query. Then the algorithm matches each tripleset of the query with each

tripleset of the database. Each triple of a tripleset of the query is matched with

each triple of the tripleset of the database. All the triples of the query set must

be unified with a triple from the database. If one triple is a rule, then unification

will use the mechanism for rules.

The modeling of a triple by owns(John, computer) is not correct because the

predicate can be a variable too.

The unication algorithm can be declared by triples and rules. It can do inferencing

about properties of graphs. A complex description of the nodes is possible

because each node can be a graph itself.

**Agents**

Agents are pieces of software that work autonomously and proactively. In most

cases, an agent will simply collect and organize information. Agents on the

Semantic Web will receive some tasks to perform and seek information from

Web resources, while communicating with other Web agents, in order to fulfill

its task. Semantic Web agents will utilize metadata, ontologies, and logic to carry

out its tasks.

**The Semantic Web and Artificial Intelligence**

Many of the technologies necessary for the Semantic Web build upon the area

of Artificial Intelligence (AI). The past difficulties in achieving AI objectives in

software applications has led to disappointment. But on the Semantic Web partial

EXERCISES **155**

solutions will work. Even if an intelligent agent is not able to come to all the

conclusions that a human counterpart could reach, the agent will still contribute

to a superior Web. The goal of the Semantic Web is to assist human users in

online activities and not to replace them.

There is no need to achieve AI at levels higher than are already available in

order to meet basic Semantic Web requirements.

**CONCLUSION**

This chapter introduced the process of forming an inference by using rules to

manipulate knowledge to produce new knowledge. In addition, we presented the

structure of inference engines and identified existing inference engines on the

Web. We discussed agents on the Semantic Web.

From this chapter, we may conclude that ontologies will play a key role

by providing vocabulary for semantic markup. Web Ontology Language is a

DL based ontology language designed for the Web that exploits the existing

standards of XML RDFS. Improved scale is necessary since reasoning is difficult

and Web ontologies may grow very large. Good empirical evidence of scalability–

tractability for conceptual reasoning with DL systems is necessary. The DLs

are a family of object-oriented KR formalisms related to frames and Semantic

networks. Descriptive Logic provides formal foundations and reasoning support.

Reasoning is important because understanding is closely related to reasoning.

Chapter 9 introduces a specific rule systems language, the Semantic Web Rule

Language.

**EXERCISES**

**8-1.** Identify the following argument’s premises and conclusions. “I think she’s

in law school; she’s always lugging around a pile of law books.”

**8-2.** Definition: An argument is valid if and only if it is absolutely impossible

that simultaneously (a) all its premises are true and (b) its conclusion is

false. Is the following argument valid: “All human beings are mortal, and

Socrates is a human being. Therefore, Socrates is mortal.” Explain.

**8-3.** Multiple choice: If the premises of an argument are true and its conclusion

is also true. Then which of the following holds: (a) The argument must be

valid. (b) The argument must be sound. (c) The argument must be valid and

sound. (d) None of the above.

**8-4.** Test the validity of the following arguments. Symbolize the propositions and

use either a truth table or an informal proof. If Mary loves cats, then John

loves dogs. John does not love dogs. Therefore Mary does not love cats.

**8-5.** Construct a truth table analysis of the following propositions: John is good

in either science or history, but not both. Moreover, either he is good at

logic or bad at history. If he is not good in science, he is bad at history. If

he is bad at history, he is good at logic.

**Figure 8-5.** Zooming in on Figure 3-2 to create a blow up of the filled in picture by a

factor 32.

**INTERLUDE #8: MACHINES AND**

**RULES**

Mary and John remained seated in the auditorium following the lecture. As other

students filed out, Mary turned to John and asked, “What makes you special?”

John said, “All humans are special, we are self-aware. We can think.”

Mary said, “Hmm. Some better than others.”

John said with amusement, “I suppose. What rules can we use to decide?”

Mary said, “Oh that reminds me, I wanted to discuss that second approach for

a ‘thought language’. I think it’s obvious that there is more to the meaning

conveyed through the combinations of words than through the individual words

examined alone. Much of the meaning comes from the relationships between the

words or concepts. In other words, rule-based systems should be considered.”

John said, “Ok. Let’s consider what constitutes a rule. If you push buttons for

2 × 4 on a calculator and get 8, you would be following a simple rule, but

how did you get the right answer? Getting the right answer is not the same as

calculating it yourself. Where was the thinking?”

Mary said, “But that’s all it takes. It is only the behavior that is important, not

how the behavior is arrived at.”

John said, “Just knowing that a rule can be mechanical like the calculator doesn’t

mean the rule the brain processes for 2 × 4 is mechanical. The crux is whether

a set of rules alone can serve to generate human cognitive behavior.”

Mary said, “Yes, we require rules as opposed to instinct.”

“Oh, but sometimes instinct can produce the right results.” John quickly interjected.

Mary continued. “Nevertheless, central to the behaviorist view of language acquisition

is learning language as a conditioned response.”

John said, “In genuine rule following, there must be a difference between actually

following a rule and appearing to follow a rule.”

Mary nodded, “How can we be certain to follow the same rules as everyone

else?”

John said, “There is an acceptance of the rules by the general public.”

**158** INTERLUDE #8: MACHINES AND RULES

Mary said, “If making judgments requires language and if language is a rulegoverned

action requiring general agreement of the rules, then I can only make

judgments or carry out thinking if there are additional intelligences?”

John said, “Yes. Then rule-following in one person necessitates rule-following

by others. But this means that a computer cannot know that it is following a rule

or program and therefore can never be thinking like a human.”

Mary said, “Well, regardless of whether language is a picturing relationship

between words and objects, or a rule-based system, I still believe thinking can be

done by a kind of thought language based upon meaningful universal symbols.”

John said, “The idea of a language of thought has problems, while words can be

interpreted by reference to what we think, my interpretation of my own thoughts

may make no sense.”

Mary said, “I think in pictures sometimes. So if thoughts do not give meaning

to sentences, they still contain symbolic meaning. But this leads to a regression

problem. For example, if I hold a letter and make a statement regarding the contents

of the letter, the sentence plus the letter is capable of fewer interpretations

than the letter alone.”

John said, “In my opinion, there are links betweens thoughts and language. For

example, the question ‘What are you thinking?’ does not elicit the thought process,

but rather the train of thought in words.”

Mary said, “Then the very process of having a thought requires the capacity to

manipulate language symbols.”

John said, “Let me summarize; thinking must be equivalent to making judgments,

but to make judgments we require language. As a result, some type of language is

mandatory for thought. And while thinking requires some symbolic representation

of real-world objects, it is not a direct manipulation of symbols in the brain as per

a set of linguistic rules. Consequently, I would suggest that there is no explicit

universal grammar of thought.”

Mary said, “I hold that pure syntax is the essence of language for encoding the

universal grammar in the brain. If this is so, then there is no distinction that

matters between this and using a computer programming language involving 0s

and 1s to replace a human language like English. The more interesting question

becomes what kind of formal system can achieve that.”

John said, “But what is the meaning of a statement in your scheme. How about

‘What color is 4?’ The grammar is correct, but its content is nonsense. The syntax

is right, but it has zero semantics.”

Mary said, “I admit I can’t answer that, just yet.”

John said, “We agree that a machine needs language as a prerequisite to think,

but we disagree on how a machine could acquire language capability.”

Mary said, “A language organ in the brain must have a universal syntax structure.”

INTERLUDE #8: MACHINES AND RULES **159**

John says, “On the other hand, meaning is the essence of language and that can

only be acquired in social context, namely, semantics.”

Mary said, “Well, how would you tell whether another human was thinking?

We can’t assess each others mental states. We must judge on the basis of that

person’s behavior. I say something, you respond. After a while I decide that you

are a thinking being.”

John said, “Thanks. From my perspective thinking requires mental states and

human life. And meaning comes from the participation in life’s experience.”

Mary said, “A computer changing symbols on a tape into new symbols is exactly

the same kind of process that the human brain goes through in the process of

thinking when it causes changes in synaptic patterns of the brain.”

John said, “So you argue, but what distinguishes humans is our ability to use

language to express new thoughts and communicate.”

Mary said, “Do you conclude that to duplicate human thought it will be necessary

for a machine to duplicate human language?”

John said, “Now there’s a thought. To pass the Turing Test, a computer must

acquire language capability.”

**9**

**SEMANTIC WEB RULE LANGUAGE**

**OVERVIEW**

The goal of incorporating logic, inference, and rules processing onto the Semantic

Web is to enable the automated use of classical perspectives on rules, and thereby

extend the reasoning capabilities of the Semantic Web. There are several rule languages

available for use on the Semantic Web including Rule Markup Language

(RuleML), Web Service Modeling Language (WSML), Semantic Web Service

Language (SWSL), and Semantic Web Rule Language (SWRL). The SWRL

specification has strong support, but this language is likely to undergo further

development, extension, and merger with features of competing technologies.

Semantic Web Rule Language is based on a combination of the OWL DL and

OWL Lite sublanguages with the sublanguages of the Rule Markup Language.

It includes a high-level abstract syntax for Horn-like rules.

Chapter 8 discussed how inference engines apply rule systems in general. This

chapter briefly describes rule languages and introduces SWRL as the likely rule

system for the Semantic Web.

**RULE SYSTEMS**

Prolog, which stands for PROgramming in LOGic, was introduced in the early

1970s and marked the beginning of rule language development. Prolog became

the most commonly used language for logic programming. Logic programming

*Thinking on the Web: Berners-Lee, G¨odel, and Turing*, by H. Peter Alesso and Craig F. Smith

Copyright 2006 John Wiley & Sons, Inc.

**161**

**162** SEMANTIC WEB RULE LANGUAGE

is based on the mathematical concepts of relations and logical inference. Prolog

consists of a database of facts and logical relationships (rules) that describe

the relationships that hold for the given application. When a user queries the

system, it searches through the database of facts and rules to determine (by

logical deduction) the answer. In cases where there is more than one solution, the

system may backtrack to generate alternative solutions. Prolog is used in Artificial

Intelligence (AI) applications, such as natural language, automated reasoning, and

expert systems.

Rules allow the expression of certain logical relationships in a form suitable

for machine processing. They include declarations like: “IF A is true, then B

must also be true.”

Software designed to interpret rules, known as “rule engines,” have become an

increasingly popular tool for implementing business rule applications for dynamic

business logic. Migration of business applications onto Web rule engines empowers

new business rule applications. These applications enable users to personalize

their preferences while empowering providers to customize their products to meet

customer needs.

There have been several efforts by industry groups to develop rule engine

capabilities to facilitate automated business practices. To date, rule engines have

been based on the use of XML, which embeds data in a formal structure with

mutually agreed upon semantic definitions. Industry initiatives using XML have

included the development of the following standards: ebXML (Electronic Business

XML Initiative), OTP (Open Trading Protocol), OBI (Open Business on the

Internet), CBL (Common Business Language), RosettaNet, eBis-XML, BizTalk,

and xCBL.

Rules may be explicitly stated or implicitly inferred. While explicit rules are

readily expressed and acted upon by rule engines, rules implicitly embedded

on the Web may not be processed even with XML. Implicit rules need to be

implemented in such a way as to allow software agents to process them.

The formal foundations of the Semantic Web allow us to infer additional

(implicit) statements that are not explicitly made. Unambiguous semantics allow

question answerers to infer that objects are the same; objects are related; or that

objects have certain restrictions. Ontologies for the Semantic Web can use rules

to define axioms operating on taxonomy. An important feature of SWRL is that it

allows us to make additional inferences beyond those provided by the ontology.

First-Order Logic (FOL) provides significant flexibility in writing down the

required axioms for a Semantic Web rule system. However, FOL sublanguages

Descriptive Logic (DL) and Horn Logic (HL) are both interesting as a rule system

because they have the properties of decidability and tractability.

**RULE LANGUAGES**

Rules have classically been used in formal languages, compiler technology,

databases, logic programming, knowledge representation, and object-oriented

SEMANTIC WEB RULE LANGUAGE **163**

modeling. Rule markup techniques for the Semantic Web, however, incorporate

rule systems (e.g., extended HL) suitable for the Web. Both derivation rules

and reaction rules are considered.

Examples of Rule languages for the Semantic Web include the Rule Markup

Language, Web Service Modeling Language, Semantic Web Service Language,

and Semantic Web Rule Language.

Rule Markup Language is based on XML, although it includes an RDF syntax.

A FOL version of RuleML is available.

Web Service Modeling Language provides an overall framework for different

logic languages. The main language paradigms supported in WSML are Description

Logics and Logic Programming.

Semantic Web Service Language is a language for describing Semantic Web

services. It has two parts: a process ontology (Semantic Web Services Ontology)

and a rules language. The rule language consists of two parts: SWSL–FOL is

a formal specification of the ontology and provides interoperability with other

first-order based process models and service ontologies. SWSL-Rules is the actual

language for service specification. Semantic Web Rule Language is a Semantic

Web language based on combining features of OWL and RuleML.

**SEMANTIC WEB RULE LANGUAGE**

The Semantic Web Rule Language was developed in 2003 by the Joint US/EU ad

hoc Agent Markup Language Committee in collaboration with RuleML Initiative,

in order to extend the expressiveness of OWL. The SWRL provides a highlevel

abstract syntax that extends the abstract syntax of OWL and uses URIs

to identify things, making it compatible with RDF and OWL. For example, in

RDF, an organization can express the fact that a particular person is an employee

who is granted access to certain information. We can use SWRL to generalize

this relationship. In SWRL, one can express the rule that being an employee

implies authorization for access to this information. Given this rule and the fact

that someone is an employee, a SWRL reasoner can conclude that the particular

person is granted information access.

Semantic Web Rule Language is an extension of OWL DL. Applications of

OWL DL can add rules to their ontologies thereby maintaining clear semantics.

Some rule systems offer meta-processing (rules about rules). The SWRL has

high expressive power, but raises questions about computational complexity for

implementation. There may be a need for selecting suitable subsets for efficient

ways to balance expressive power against execution speed and termination of the

computation.

An OWL ontology contains a sequence of axioms and facts. These axioms may

include a variety of kinds (e.g., subClass axioms and equivalentClass axioms).

However, rule axioms can be extended.

The rules form an implication between antecedent (body) and consequent

(head). The intended meaning implies whenever the conditions specified in the

**164** SEMANTIC WEB RULE LANGUAGE

antecedent hold, then the conditions in the consequent must also hold (see

Chapter 8).

Both the antecedent and consequent consist of atoms or elementary statements.

An empty antecedent is treated as trivially true; an empty consequent is treated

as trivially false. Multiple atoms are treated as a conjunction. The rules with

conjunctive consequents can be transformed into multiple rules each with an

atomic consequent.

Atoms in rules can be of the form C(*x*), P(*x*,*y*), sameAs(*x*,*y*) or differentFrom(

*x*,*y*), where C is an OWL description, P is an OWL property, and

*x*,*y* are either variables, OWL individuals or OWL data values. However, OWL

DL becomes undecidable when extended in this way.

An XML syntax is also given for these rules based on RuleML. Atoms may

refer to individuals, data literals, individual variables, or data variables. Variables

are treated as universally quantified, with their scope limited to a given rule.

Using this syntax, a rule asserting that the composition of parent and brother

properties implies the uncle property would be written as:

<ruleml:imp>

<ruleml: body>

<swrlx:individualPropertyAtom swrlx:property="hasParent">

<ruleml:var>x1</ruleml:var>

<ruleml:var>x2</ruleml:var>

</swrlx:individualPropertyAtom>

<swrlx:individualPropertyAtom swrlx:property="hasBrother">

<ruleml:var>x2</ruleml:var>

<ruleml:var>x3</ruleml:var>

</swrlx:individualPropertyAtom>

</ruleml: body>

<ruleml: head>

<swrlx:individualPropertyAtom swrlx:property="hasUncle">

<ruleml:var>x1</ruleml:var>

<ruleml:var>x3</ruleml:var>

</swrlx:individualPropertyAtom>

</ruleml: head>

</ruleml:imp>

A simple use of these rules would be to assert that the combination of the

hasParent and hasBrother properties implies the hasUncle property. From this

rule, if John has Mary as a parent, and Mary has Bill as a brother, then John has

Bill as an uncle.

**CONCLUSION**

This chapter introduced Semantic Web Rule Language (SWRL). The SWRL is

based on a combination of the OWL DL and OWL Lite sublanguages with the

**SEMANTIC WEB APPLICATIONS**

**OVERVIEW**

Today, computers and small devices are being used to access, from any location,

an ever-increasing flood of Web information. As the size of the Web expands, and

with it its information content, it is becoming more and more difficult to search,

access, maintain, and manage network resources. Creating machine-processable

semantics could alleviate some of these difficulties. The resulting Semantic Web

applications could provide intelligent access to heterogeneous, distributed information,

enabling software products (and agents) to mediate between user needs

and the information sources available.

This chapter, describes some of the application areas for semantic technology.

We focus on ongoing work in the fields of knowledge management and

electronic commerce. Some opportunities for Semantic Web applications include

Semantic Web Services, Semantic Search, e-Learning, Semantic Web and Bio-

Informatics, Semantics-based Enterprise application and data integration, and

Knowledge Base.

**SEMANTIC WEB APPLICATIONS**

Semantic Web applications are those web-based applications that take advantage

of semantic content: content that includes not only information, but also

*Thinking on the Web: Berners-Lee, G¨odel, and Turing*, by H. Peter Alesso and Craig F. Smith

Copyright 2006 John Wiley & Sons, Inc.

**169**

**170** SEMANTIC WEB APPLICATIONS

metadata, or information about information. The Semantic Web can be used

for more effective discovery, automation, integration, and reuse across various

applications.

The Semantic Web will provide an infrastructure not just for Web pages, but

databases, services, programs, sensors, and personal devices. Software agents can

use this information to search, filter, and repackage information. The ontology

and logic languages will make information machine readable and power a new

generation of tools.

Web technologies can link information easily and seamlessly. The majority of

network systems now have Web servers, and the Web interfaces make them seem

part of the same world of information. Despite this, transferring content between

Web applications is still difficult.

The Semantic Web can address and improve the linking of databases, sharing

content between applications, and discovery and combination of Web Services.

Under the current Web architecture, linkages between dissimilar systems are

provided by costly, tailored software. Again and again, special purpose interfaces

must be written to bring data from one systems into another. Applications that run

in a given company involve a huge number of ways they can be linked together.

That linking requires a lot of custom code. Use of XML can help, but the problem

of effectively exchanging data remains. For every pair of applications someone

has to create an “XML to XML bridge.”

The problem is that different databases are built using different database

schemas, but these schemas are not made explicit. Just as older database systems

suddenly became compatible by adopting a consistent relational model, so

unstructured Web data, or XML schema definitions, can adopt a relational model.

The use of Resource Description Framework (RDF) in addition to XML

can be appropriate when information from two sources need to be merged or

interchanged. It is possible to concatenate the files joining on defined terms to

correspond to the same Universal Resource Indicators (URIs). When you want to

extend a query on one RDF file to include constraints from another, you just add

in the constraints as part of the merging. Where XML is made up of elements and

attributes, RDF data is made up of statements where each statement expresses

the value of one property.

The Semantic Web is bringing to the Web a number of capabilities, such

as allowing applications to work together in a decentralized system without a

human having to custom handcraft every connection. The business market for

this integration of data and programs is huge, and we believe the companies who

choose to start exploiting Semantic Web technologies will be the first to reap the

rewards.

Some opportunities for Semantic Web applications include Semantic Web

Services, Semantic Search, e-Learning, Semantic Web and Bio-Informatics, Semantics-

based Enterprise Application and Data Integration, and Knowledge Base.

We will discuss these in the following sections.

SEMANTIC SEARCH **171**

**SEMANTIC WEB SERVICES**

Semantic Web Services can bring programs and data together. Just as databases

cannot be easily integrated on the current Web without RDF, the same applies to

programs. Unfortunately, many e-business applications particularly in businessto-

business (B2B) interactions have difficulty loading someone else’s program to

run locally.

Consider the case of a company that wishes to purchase parts from a vendor,

arrange shipping from a large freight company, and have those parts delivered to

one of several manufacturing locations based on which plant has the most capacity

at the time of the delivery. Further, they would like this deal to be brokered on

the Web with the minimum amount of human interaction. These programs that

execute this brokering may be running on special purpose machines and/or behind

security and firewall protections. How can all these programs interoperate on the

Web to provide protocols and descriptions of the “services” that these various

programs offer?

Web Services are self-contained, self-described, component applications

invoked across the Web to perform complex business processes. Once a Web

Service is deployed, other applications can discover and invoke the service.

At present, Web Services require human interaction in order to identify and

implement.

Tim Berners-Lee has suggested that the integration of Web Services and the

Semantic Web could be done in such a way as to combine the business logic

of Web Services with the Semantic Web’s meaningful content. There are several

areas where the current technologies for discovery (UDDI or Universal Description,

Discovery, and Integration), binding (WSDL or Web Services Description

Language), and messaging (SOAP or Simple Object Access Protocol) could use

OWL to provide an ontology for automatic SemanticWeb Services thereby allowing

greater interaction with Web business rules’ engines.

The vision for Semantic Web Services is to automate the discovery, invocation,

composition, and monitoring of Web Services through the use of machine

processing. Web sites will be able to use a set of classes and properties by declaring

and describing an ontology of services. Web Ontology Language for Services

(called OWL-S) has been designed to meet this goal. Semantic Web Services and

OWL-S will be described, in greater detail in Chapter 11.

**e-LEARNING**

The big question in the area of educational systems is what is the next step in

the evolution of e-learning? Are we finally moving from scattered applications

to a coherent collaborative environment? How close we are to the vision of the

Educational Semantic Web and what do we need to do in order to realize it?

On the one hand, we wish to achieve interoperability among educational systems

and on the other hand, to have automated, structured, and unified authoring.

The Semantic Web is the key to enabling the interoperability by capitalizing on

(*1*) semantic conceptualization and ontologies, (*2*) common standardized communication

syntax, and (*3*) large-scale integration of educational content and usage.

The RDF describes objects and their relationships. It allows easy reuse of

information for different devices, such as mobile phones and PDAs, and for

presentation to people with different capabilities, such as those with cognitive or

visual impairments.

It is possible that in the near future students will be able to extract far more

data from a networked computer or wireless device, far more efficiently. Based

on a few specific search terms, library catalogues could be scanned automatically

and nearest library shelf marks delivered immediately to students, alongside multimedia

and textual resources culled from the Web itself. Students could also be

directed to relevant discussion lists and research groups.

By tailored restructuring of information, future systems will be able to deliver

content to the end-user in a form applicable to them, taking into account users’

needs, preferences, and prior knowledge. Much of this work relies on vast online

databases and thesauri, such as wordnet, which categorize synonyms into distinct

lexical concepts. Developing large multimedia database systems makes materials

as useful as possible for distinct user groups, from schoolchildren to university

lecturers. Students might, therefore, search databases using a simple term, while

a lecturer might use a more scientific term thus reflecting scaling in complexity.

The educational sector can also use the Internet Relay Chat (IRC)

(http://www.irc.org/) a tool that can be used by the Semantic Web. The IRC

ENTERPRISE APPLICATION INTEGRATION **173**

is a chat protocol where people can meet on channels and talk to each other. The

Semantic Web community is enhancing this capability by writing robots that can

help to log the chat when members are away. It can also assist with meetings,

discussions, and recording of results.

The IRC and related tools could work well within education, for project discussion,

remote working, and collaborative document creation. Video-conferencing

at schools is increasingly becoming useful in widening the boundaries for students.

The incorporation of Semantic Web technologies could create the ability to

work across distributed locations in communities of learning and enable content

creation outside of the classroom.

**Development of Social Network Analysis**

The field of Social Network Analysis today is the result of the convergence of several

streams of applied research in sociology, social psychology and anthropology.

Many of the concepts of network analysis have been developed independently

by various researchers often through empirical studies of various social settings. For

example, many social psychologists of the 1940s found a formal description of social

groups useful in depicting communication channels in the group when trying to explain

processes of group communication. Already in the mid-1950s anthropologists

have found network representations useful in generalizing actual field observations,

for example when comparing the level of reciprocity in marriage and other social

exchanges across different cultures.

Some of the concepts of network analysis have come naturally from social studies.

In an influential early study at the Hawthorne works in Chicago, researchers from

Harvard looked at the workgroup behavior (e.g. communication, friendships, helping,

controversy) at a specific part of the factory, the bank wiring room [May33].

The investigators noticed that workers themselves used specific terms to describe

who is in “our group”. The researchers tried to understand how such terms arise by

reproducing in a visual way the group structure of the organization as it emerged

from the individual relationships of the factory workers (see Figure 2.1).2 In another

study of mixed-race city in the Southern US researchers looked at the network of

overlapping “cliques” defined by race and age [WL41].3 They also went further than

the Hawthorne study in generating hypotheses about the possible connections between

cliques. (For example, they noted that lower-class members of a clique are

usually only able to connect to higher-class members of another clique through the

higher-class members of their own clique.)

Despite the various efforts, each of the early studies used a different set of concepts

and different methods of representation and analysis of social networks. However,

from the 1950s network analysis began to converge around the unique world

view that distinguishes network analysis from other approaches to sociological research.

(The term “social network” has been introduced by Barnes in 1954.) This

convergence was facilitated by the adoption of a graph representation of social networks

usually credited to Moreno. What Moreno called a *sociogram* was a visual

representation of social networks as a set of nodes connected by directed links. The

nodes represented individuals in Moreno’s work, while the edges stood for personal

relations. However, similar representations can be used to depict a set of relationships

2 The study became famous not so much of the network methods used but for what became

known in management science as the *Hawthorne-effect*. In brief, managers at the

Hawthorne factory were initially trying to understand what alterations in the work conditions

affect productivity. To their surprise no matter what the change was it seemed to affect

productivity in a positive way. Mayo and colleagues concluded that the mere participation

in the research project itself was the key factor as workers were pleased with the management

taking an interest in their conditions. Although it became widely known, the original

study as well as the general existence of this effect is disputed [Gil93].

3 Clique is a term that now has a precise definition in network analysis.

30 2 Social Network Analysis

**Figure 2.1.** Illustrations froman early social network study at the Hawthorne works ofWestern

Electric in Chicago. The upper part shows the location of the workers in the wiring room, while

the lower part is a network image of fights about the windows between workers (W), solderers

(S) and inspectors (I).

between any kind of social unit such as groups, organizations, nations etc. While 2D

and 3D visual modelling is still an important technique of network analysis, the sociogram

is honored mostly for opening the way to a formal treatment of network

analysis based on graph theory.

The following decades have seen a tremendous increase in the capabilities of

network analysis mostly through new applications. SNA gains its relevance from

applications and these settings in turn provide the theories to be tested and greatly

influence the development of the methods and the interpretation of the outcomes.

For example, one of the relatively new areas of network analysis is the analysis of

networks in entrepreneurship, an active area of research that builds and contributes

to organization and management science.

2.3 Key concepts and measures in network analysis 31

The vocabulary, models and methods of network analysis also expand continuously

through applications that require to handle ever more complex data sets. An

example of this process are the advances in dealing with longitudinal data. New probabilistic

models are capable of modelling the evolution of social networks and answering

questions regarding the dynamics of communities. Formalizing an increasing

set of concepts in terms of networks also contributes to both developing and

testing theories in more theoretical branches of sociology.

The increasing variety of applications and related advances in methodology can

be best observed at the yearly Sunbelt Social Networks Conference series, which

started in 1980.4 The field of Social Network Analysis also has a journal of the same

name since 1978, dedicated largely to methodological issues.5 However, articles describing

various applications of social network analysis can be found in almost any

field where networks and relational data play an important role.

While the field of network analysis has been growing steadily from the beginning,

there have been two developments in the last two decades that led to an explosion

in network literature. First, advances in information technology brought a

wealth of electronic data and significantly increased analytical power. We examine

the possibilities of using electronic data for network analysis in Chapter 3. Second,

the methods of SNA are increasingly applied to networks other than social networks

such as the hyperlink structure on the Web or the electric grid. This advancement

—brought forward primarily by physicists and other natural scientists— is based on

the discovery that many networks in nature share a number of commonalities with

social networks. In the following, we will also talk about networks in general, but

it should be clear from the text that many of the measures in network analysis can

only be strictly interpreted in the context of social networks or have very different

interpretation in networks of other kinds.

**Electronic sources for network analysis**

From the very beginning of the discipline collecting data on social networks required

a certain kind of ingenuity from the researcher. First, social networks have been

studied by observation. The disadvantage of this method is the close involvement

of the researcher in the process of data collection. Standardized surveys minimize

(but do not completely eradicate) the influence of the observer but they rely on an

active engagement of the population to be studied. Unfortunately, as all of us are

flooded these days by surveys of all kinds, achieving a high enough response rate

for any survey becomes more and more problematic. In some settings such as within

companies surveys can be forced on the participants, but this casts serious doubts

on whether the responses will be spontaneous and genuine. Worse yet, observations

and surveys need to be repeated multiple times if one would like to study network

dynamics in any detail.

Data collection using these manual methods are extremely labor intensive and

can take up to fifty per cent of the time and resources of a project in network analysis.

Oftentimes the effort involved in data collection is so immense that network

researchers are forced to reanalyze the same data sets over and over in order to be

able to contribute to their field.

Network analysts looking for less costly empirical data are often forced to look

for alternatives. A creative solution to the problem of data collection is to reuse existing

electronic records of social interaction that were not created for the purposes

of network analysis on the first place. Scientific communities, for example, have

been studied by relying on publication or project databases showing collaborations

among authors or institutes [BJN+02, GM02]. Official databases on corporate technology

agreements allow us to study networks of innovation [Lem03], while newspaper

archives are a source of analysis for studies on topics ranging from the role

of social-cognitive networks in politics [vAKOS06] to the structure of terror organizations

[Kre02]. These sources often support dynamic studies through historical

analysis. Nevertheless, the convenience comes at a price: access to publication and

patent databases, media archives, legal and financial records often carries a significant

price tag.

52 3 Electronic sources for network analysis

However, there is one data source that is not only vast, diverse and dynamic

but also free for all: the Internet. In the following, we look at a sample of works

from the rapidly emerging field of *e-social science*. Common to these studies is that

they rely entirely on data collected from electronic networks and online information

sources, which allows a complete automation of the data collection process. None

of these works rely on commercial databases and yet many of them are orders of

magnitude larger than studies based on data collected through observation or surveys.

They represent a diversity of social settings and a number of them also exploit the

dynamics of electronic data to perform longitudinal analysis. We will spend more

attention on methods of social network extraction from the Web that we use in our

analysis of the SemanticWeb community (Chapter 8).

There are limits of course to the potential of *e-social science*. Most trivially,

what is not on the Web can not be extracted from the Web, which means that there

are a number of social settings that can only be studied using offline methods. There

also technological limits to the accuracy of any method that relies on Information

Extraction. For these reasons it is natural to evaluate our methods before using them

for network analysis. We return to this issue in Chapter 7.

**3.1 Electronic discussion networks**

One of the foremost studies to illustrate the versatility of electronic data is a series

of works from the Information Dynamics Labs of Hewlett-Packard.

Tyler, Wilkinson and Huberman analyze communication among employees of

their own lab by using the corporate email archive [TWH03]. They recreate the actual

discussion networks in the organization by drawing a tie between two individuals

if they had exchanged at least a minimum number of total emails in a given period,

filtering out one-way relationships. Tyler et al. find the study of the email network

useful in identifying leadership roles within the organization and finding formal as

well as informal communities. (Formal communities are the ones dictated by the organizational

structure of the organization, while informal communities are those that

develop across organizational boundaries.) The authors verify this finding through a

set of interviews where they feed back the results to the employees of the Lab.

Wu, Huberman, Adamic and Tyler use this data set to verify a formal model

of information flow in social networks based on epidemic models [WHAT04]. In

yet another study, Adamic and Adar revisits one of the oldest problems of network

research, namely the question of *local search*: how do people find short paths in

social networks based on only local information about their immediate contacts?

Their findings support earlier results that additional knowledge on contacts such as

their physical location and position in the organization allows employees to conduct

their search much more efficiently than using the simple strategy of always passing

the message to the most connected neighbor. Despite the versatility of such data, the

studies of electronic communication networks based on email data are limited by

privacy concerns. For example, in the HP case the content of messages had to be

ignored by the researchers and the data set could not be shared with the community.

3.2 Blogs and online communities 53

Public forums and mailing lists can be analyzed without similar concerns. Starting

from the mid-nineties, Marc Smith and colleagues have published a series of

papers on the visualization and analysis of USENET newsgroups, which predate

Web-based discussion forums (see the author’s homepage or the book [Smi99]). In

the work of Peter Gloor and colleagues, the source of these data for analysis is the

archive of the mailing lists of a standard setting organization, the World Wide Web

Consortium (W3C) [GLDZ03]. The W3C —which is also the organization responsible

for the standardization of Semantic Web technologies—is unique among standardization

bodies in its commitment to transparency toward the general public of

the Internet and part of this commitment is the openness of the discussions within

the working groups. (These discussion are largely in email and to a smaller part on

the phone and in face-to-face meetings.)

Group communication and collective decision taking in various settings are traditionally

studied using much more limited written information such as transcripts

and records of attendance and voting, see e.g. As in the case with emails Gloor uses

the headers of messages to automatically re-create the discussion networks of the

working group.1 The main technical contribution of Gloor is a dynamic visualization

of the discussion network that allows to quickly identify the moments when key

discussions take place that activate the entire group and not just a few select members.

Gloor also performs a comparative study across the various groups based on

the structures that emerge over time.

Although it has not been part of this work, it would be even possible to extend

such studies with an analysis of the role of networks in the decision making process

as voting records that are also available in electronic formats. Further, by applying

emotion mining techniques from AI to the contents of the email messages one could

recover agreements and disagreements among committee members. Marking up the

data set manually with this kind of information is almost impossible: a single working

group produces over ten thousand emails during the course of its work.

**3.2 Blogs and online communities**

Content analysis has also been the most commonly used tool in the computer-aided

analysis of blogs (web logs), primarily with the intention of trend analysis for the

purposes of marketing.2 While blogs are often considered as “personal publishing”

or a “digital diary”, bloggers themselves know that blogs are much more than that:

modern blogging tools allow to easily comment and react to the comments of other

1 A slight difference is that unlike with personal emails messages to a mailing list are read

by everyone on the list. Nevertheless individuals interactions can be partly recovered by

looking at To: and CC: fields of email headers as well as the Reply-To field.

2 See for example the works presented at the 2006 AAAI Spring Symposium on Computational

Approaches to Analyzing Weblogs at http://www.umbriacom.com/aaai2006\_

weblog\_symposium/ or the Workshop on the Weblogging Ecosystem at http://

wwe2005.blogspot.com/

54 3 Electronic sources for network analysis

bloggers, resulting in webs of communication among bloggers. These discussion networks

also lead to the establishment of dynamic communities, which often manifest

themselves through syndicated blogs (aggregated blogs that collect posts from a set

of authors blogging on similar topics), blog rolls (lists of discussion partners on a

personal blog) and even result in real world meetings such as the Blog Walk series

of meetings3. Figure 3.1 shows some of the features of blogs that have been used in

various studies to establish the networks of bloggers.

Comments

Links from

other blogs

Link to

another blog

post

Quote

Link to

another blog

**Figure 3.1.** Features of blogs that can be used for social network extraction. Note also that

—unlike web pages in general—blog entries are timestamped, which allows to study network

dynamics, e.g. the spread of information in online communities.

Blogs make a particularly appealing research target due to the availability of

structured electronic data in the form of RSS (Rich Site Summary) feeds. RSS feeds

contain the text of the blog posts as well as valuable metadata such as the timestamp

of posts, which is the basis of dynamic analysis. For example, Kumar et al. and

Gruhl et al. study information diffusion in blogs based on this information [KNRT03,

GGLNT04]. The early work of Efimova and Anjewierden also stands out in that they

were among the first to study blogs from a communication perspective [AE06]. Adar

and Adamic offer a visualization of such communication in blogs in [AA05b].

The 2004 US election campaign represented a turning point in blog research

as it has been the first major electoral contest where blogs have been exploited as

a method of building networks among individual activists and supporters (see for

example [AG05]). Blog analysis has suddenly shed its image as relevant only to

3 http://blogwalk.interdependent.biz/wikka.php?wakka=HomePage

3.3 Web-based networks 55

marketers interested in understanding product choices of young demographics; following

this campaign there has been explosion in research on the capacity of web

logs for creating and maintaining stable, long distance social networks of different

kinds. Since 2004, blog networks have been the object of study for a number of

papers in the blog research track of the yearly Sunbelt social networks conference.

Online community spaces and social networking services such as MySpace,

LiveJournal cater to socialization even more directly than blogs with features such

as social networking (maintaining lists of friends, joining groups), messaging and

photo sharing.4 As they are typically used by a much younger demographic they

offer an excellent opportunity for studying changes in youth culture. Paolillo, Mercure

and Wright offer a characterization of the LiveJournal community based on

the electronic data that the website exposes about the interests and social networks

of its users [PMW05]. Backstrom et al. also study the LiveJournal data in order to

answer questions regarding the influence of certain structural properties on community

formation and community growth, while also examining how changes in the

membership of communities relates to (changes in) the underlying discussion topics

[BHKL06]. These studies are good examples of how directly available electronic

data enables the longitudinal analysis of large communities (more than 10,000 users).

Similar to our work in Chapter 8 these studies also go beyond investigating purely

structural network properties: in posing their questions they build on the possibility

to access additional information about user interests.

LiveJournal exposes data for research purposes in a semantic format, but unfortunately

this is the exception rather than the norm. Most online social networking

services (Friendster, Orkut, LinkedIn and their sakes) closely guard their data even

from their own users. (Unless otherwise stated these data provided to an online service

belongs to the user. However, most of these services impose terms of use that

limit the rights of their users.) A technological alternative to these centralized services

is the FOAF network (see also Chapter 5). FOAF profiles are stored on the web

site of the users and linked together using hyperlinks. The drawback of FOAF is that

at the moment there is a lack of tools for creating and maintaining profiles as well as

useful services for exploiting this network. Nevertheless, a few preliminary studies

have already established that the FOAF network exhibits similar characteristics to

other online social networks [PW04, DZFJ05].

**6.1 Building Semantic Web applications with social network**

**features**

In the following we first sketch the shared design of most current Semantic Web

application. This will help us to pinpoint the focus of Semantic Web application

development, and the role of triple stores and ontology APIs.

Next, we introduce Sesame3, a general database for the storing and queryingRDF

data. Along with Jena4, Redland5 and the commercial offerings of Oracle6, Sesame is

one of the most popular triple stores among developers, appreciated in particular for

its performance. Sesame has been developed by Aduna (formerly Aidministrator),

but available as open source (currently under LGPL license).

Next, we describe the Elmo API, a general purpose ontology API for Sesame.

Elmo allows to manipulate RDF/OWL data at the level of domain concepts, with

specific tools for collecting and aggregating RDF data from distributed, heterogeneous

information sources. Elmo has been developed in part by the author and is

available under the same conditions as Sesame, using the same website.

Lastly, we introduce a simple utility called GraphUtil which facilitates reading

FOAF data into the graph object model of the Java UniversalNetwork Graph (JUNG)

API. GraphUtil is open source and available as part of Flink (see Section 6.2).

3 http://www.openrdf.org

4 http://jena.sourceforge.net/

5 http://librdf.org/

6 http://www.oracle.com/technology/tech/semantic\_technologies/index.

html

124 6 Developing social-semantic applications

**6.1.1 The generic architecture of SemanticWeb applications**

As the history of submissions to the Semantic Web challenge attest, Semantic Web

applications have been developed in the past years in a wide range of domains from

cultural heritage to medicine, from music retrieval to e-science. Yet, almost all share

a generic architecture as shown in Figure 6.1. By the definition above, all Semantic

Web applications are mashups in that they build on a number of heterogeneous data

sources and services under diverse ownership or control.

Before external, heterogeneous data sources can be reused, they need to be normalized

syntactically as well as semantically. The first refers to transforming data

into an RDF syntax such as RDF/XML, while the latter means that the ontologies

(schemas and instances) of the data sources need to be reconciled. Needless to say,

the first step can be skipped if the data is exposed as an RDF or OWL document, or

can be queried dynamically using the SPARQL query language and protocol.

Sesame storage and reasoning

Sesame Client API

Sesame

servlets

Elmo

Application

SPARQL

HTTP

SOAP

JavaBeans

Source 1

Source 2

Source N

Triples

**Figure 6.1.** The generic design of SemanticWeb applications using Sesame and Elmo. Developing

with other triple stores results in similar architectures, but in general application code is

not portable among triple stores due to proprietary APIs.

Most current Semantic Web applications are based on a fixed, small number of

data sources selected by the application developer. In this case, the schemas of the

data sources are known in advance and their mapping can be performed manually. In

6.1 Building Semantic Web applications with social network features 125

the future, it is expected that SemanticWeb applications will be able to discover and

select new data sources and map them automatically.7

Semantic Web applications persist information in ontology stores (also known

as triple stores), databases specifically designed for the storage, manipulation and

querying of RDF/OWL data. Ontology stores are almost always equipped with a

reasoner or can be connected to an external reasoning service. Reasoning is used to

infer new information based on the asserted facts or to check the existing information

for consistency. Some triple stores also allow to define customrules that are evaluated

by the reasoner along with the rules prescribed by the ontology language itself. As we

have discussed in Chapter 5, the task of instance unification can also be partly solved

by OWL DL or rule-based reasoning. Reasoning can take place either when the data

is added to a repository (forward-chaining) or at query time (backward-chaining).

Most Semantic Web applications have a web interface for querying and visualization

and thus considered by all as web applications. However, this is not a requirement:

Semantic Web applications may have a rich client interface (desktop applications)

or other forms of access. More importantly, Semantic Web applications are

expected to expose data in the same way they expect to consume the data of other

applications: using the standard languages and protocols of the Semantic Web, and

conforming to the architectural style of theWeb in general8 In order to facilitate this,

most triple stores implement the SPARQL query language and protocol (see Section

4.2.1) and some also implement REST9 style interfaces. A SPARQL service allows

other applications to query the triple store, but it provides no data manipulation

features such as adding or removing data. Therefore most triple stores also provide

custom web interfaces for data manipulation. A Semantic Web application may also

expose data or services at higher levels of abstraction than the level of triples, i.e. on

the level of domain objects and operations that can be executed on them.

As one would assume, the application logic of Semantic Web applications is

placed between the triple store and the eventual web interface(s). The application

normally accesses the triple store through its client API.When working with the API

of the triple store, the programmer manipulates the data at the level of RDF triples,

7 Regrettably, one of the missing pieces of the Semantic Web infrastructure is an agreed

way of publishing ontologies and metadata online. Finding ontologies and RDF/OWL data

sets requires one to either search using a traditional search engine or to query a search

engine targeted at indexing ontologies, such as Swoogle[DFJ+04]. However, neither of

these methods provide adequate meta-information about the ontologies, for example by

whom they have been developed, for what purpose and what web applications are using

them. Another stumbling block in the way of automatically integrating new data sources

is the difficulty of automating schema mapping. Automated ontology (schema) mapping is

among the hardest research problems and the methods that exist are certainly not on par

with the performance of humans carrying out these tasks. Nevertheless, automated methods

for ontology mapping are crucial for this future scenario, where only computers will be able

to cope with the amount of schema information to be processed.

8 By respecting, for example, the guidelines for linking data, see http://www.w3.org/

DesignIssues/LinkedData.html.

9 Representational State Transfer

126 6 Developing social-semantic applications

i.e. the basic operations are adding and removing triples. (Updates are handled by

removing the given triple and then adding a new one.) Queries are given as a combination

of triple patterns and return a table (a set of variable bindings) as a result.

This is similar to accessing a relational database. Its notable, however, that at the

current stage of developments applications can only access triple stores through proprietary

APIs or the above mentioned SPARQL protocol, which provides limited,

read-only access and is only suitable for accessing remote data sources. (With most

web applications, the database and the application are co-hosted on the computer.

In such a situation communicating over HTTP adds unnecessary overhead.) In other

words, what is lacking is an equivalent of the ODBC and JDBC protocols for relational

databases. This means that without additional abstraction layers (such as the

one provided by Elmo), all application code is specific to a particular triple store.

Further, in most cases it is desirable to access a triple store on an ontological level,

i.e. at the level of classes, instances and their properties. This is also the natural level

of manipulating data in object-oriented frameworks. The Elmo library to be introduced

facilitates this by providing access to the data in the triple store through Java

classes that map the ontological data in the triple store. Setting and reading attributes

on the instances of these classes result in adding and removing the corresponding

triples in the data store.

Elmo is a set of interfaces that have been implemented for the specific case of

working with data in Sesame triple stores. Sesame is one of the most popular RDF

triple stores and it is to be introduced next. We note that the Elmo interfaces can

be implemented for other, Java-based triples stores such as Jena. Interfacing with

non-Java triple stores would require an agreement on standard protocols similar to

JDBC.

**6.1.2 Sesame**

Sesame is a triple store implemented using Java technology. Much like a database

for RDF data, Sesame allows creating repositories and specifying access privileges,

storing RDF data in a repository and querying the data using any of the supported

query languages. In the case of Sesame, these include Sesame’s own SeRQL language

and SPARQL. (While SPARQL has the advantage in terms of standardization,

it is also minimal by design; SeRQL is a more expressive query language with many

useful features.) The data in the repository can be manipulated on the level of triples:

individual statements can be added and removed from the repository. (There is no

direct update operation. Updates can be carried out by removing and then adding a

statement.) RDF data can be added or extracted in any of the supported RDF representations

including the RDF/XML and Turtle languages introduced in Chapter 4.

Sesame can persistently store and retrieve the data from a variety of back-ends: data

can persist in memory, on the disk or in a relational database.

As most RDF repositories, Sesame is not only a data store but also integrates reasoning.

Sesame has a built-in inferencer for applying the RDF(S) inference rules (see

Section 4.2.1). While Sesame does not support OWL semantics, it does have a rule

language that allows to capture most of the semantics of OWL, including the notion

6.1 Building Semantic Web applications with social network features 127

of inverse-functional properties and the semantics of the *owl*:*sameAs* relationship

(see Section 5.4.2). Reasoning can be enabled or disabled for specific repositories.

When enabled, reasoning is performed at the time when data is added to the repository

or when it is removed10

An important, recently added feature of Sesame is the ability to store and retrieve

context information. In distributed scenarios, it is often necessary to capture metadata

about statements. For example, in the case of collecting FOAF profiles from the

Web, we might want to keep track of where the information came from (the URL of

the profile) and the time it was last crawled. Context information is important even

for centralized sites with user contributed content. In previous versions of Sesame,

the only possibility to store context information was to represent it using the reification

mechanism of RDF (see Section 4.2.1), which is very inefficient. Starting from

Sesame 2.0, the repository natively supports the storage and querying of context information.

In effect, every triple becomes a *quad*, with the last attribute identifying

the context. Contexts are identified by resources, which can be used in statements as

all other resources. Contexts (named graphs) can also be directly queried using the

SPARQL query language supported by this version of Sesame.

The above mentioned functionalities of Sesame can be accessed in three ways.

First, Sesame provides an HTML interface that can be accessed through a browser.

Second, a set of servlets exposes functionality for remote access through HTTP,

SOAP and RMI. Lastly, Sesame provides a Java client library for developers which

exposes all the above mentioned functionality of a Sesame repository using method

calls on a Java object called *SesameRepository*. This object can provide access to

both local Sesame servers (running in the same Java Virtual Machine as the application)

or and remote servers (running in a different JVM as the application or on a

remote machine.

Working with the Sesame client API is relatively straightforward. Queries, for

example, can be executed by calling the *evaluateTableQuery* method of this class,

passing on the query itself and the identifier of the query language. The result is

another object (*QueryResultsTable*) which contains the result set in the form of a

table much like the one shown in the web interface (see Figures 6.2 and 6.3). Every

row is a result and every column contains the value for a given variable. The values

in the table are objects of type *URI*, *BNode* or *Literal*, the object representations

of the same notions in RDF. For example, one may call the *getValue*, *getDatatype*

and *getLanguage* methods of *Literal* to get the String representation of the literal, its

datatype and its language.

Sesame’s client library is appropriate for manipulating RDF data at the level

of individual triples. Object-oriented applications, however, manipulate data at the

level of objects and their attributes; and while objects are characterized by a set of

attributes and their values, individual triples capture only a single value for a single

property. Updating an attribute of an object may translate to updating several triples.

Similarly, removing an object, results in the removal of a number of triples. There

10 When a statement is removed, one also needs to remove in addition all those statements

that were inferred from the removed statement (and from that statement only).

128 6 Developing social-semantic applications

**Figure 6.2.** Adding data to a Sesame repository using the web interface. Here we add the data

used in the examples of Chapter 4.

is thus a need for an API that can translate between operations on objects and the

underlying triple representation. This is one of the main concerns of the Elmo API.

**6.1.3 Elmo**

Elmo is a development toolkit consisting of two main components. The first one is the

Elmo API, providing the above mentioned interface between a set of JavaBeans representing

ontological classes and the underlying triple store containing the data that

is manipulated through the JavaBeans. The API also includes the tool for generating

JavaBeans from ontologies and vice versa. The second main component consists of a

set of tools for working with RDF data, including an RDF crawler and a framework

of smushers (instance unification methods).

6.1 Building Semantic Web applications with social network features 129

**Figure 6.3.** Querying data through the web interface of Sesame. The results are displayed

below in a table.

**The Elmo API**

The core of the Elmo API is the *ElmoManager* a JavaBean pool implementation that

is responsible for creating, loading, renaming and removing ElmoBeans. ElmoBeans

are a composition of *concepts* and *behaviors*. Concepts are Java interfaces that correspond

one-to-one to a particular ontological class and provide getter and setter

methods corresponding to the properties of the ontological class. (The mapping is

maintained using annotations on the interface.) The inheritance hierarchy of the ontological

classes is mapped directly to the inheritance hierarchy of concepts. Elmo

concepts are typically generated using a code-generator.

Instances of ElmoBeans corresponds to instances of the data set. As resources in

an ontology may have multiple types, ElmoBeans themselves need to be composed

of multiple concepts. ElmoBeans implement particular combinations of concept interfaces.

Note that it is necessary to generate separate classes for every particular

combination of types that are occurring in the data set, because its not possible in

130 6 Developing social-semantic applications

Java for an instance to belong to multiple classes. ElmoBeans can be generated runtime

as the types of resources may change during the run-time of the application.

ElmoBeans may also implement behaviors. Behaviors are concrete or abstract

classes that can be used to give particular implementations of the methods of a concept

(in case the behavior should differ from the default behavior), but can also be

used to add additional functionality. Behaviours can be mixed-in to ElmoBeans the

same way that additional types can be added runtime.

The separation of concepts and behaviors, and the ability to compose them at

will supports the distributed application development, which is the typical scenario

in case of Web applications.

As a separate package, Elmo also provides ElmoBean representations for the

most popularWeb ontologies, including FOAF, RSS 1.0 and Dublin Core. For example,

in the FOAF model there is *Person* JavaBean with the properties of *foaf* :*Person*.

Getting and setting these properties manipulates the underlying RDF data. This

higher level of representation significantly simplifies development. For example, a

simple FOAF profile can be created in ten lines of Java code (see Figure 6.4).

Repository repository = **new** SailRepository(**new** MemoryStore());

repository.initialize();

SesameManagerFactory factory =

**new** SesameManagerFactory(repository);

ElmoManager manager = factory.createElmoManager();

QName jackID = **new** QName("http://www.example.org#","jack");

Person jack = manager.createBean(jackID, Person.**class**);

jack.getFoafFirstName().add("Jack");

System.out.println(jack.getFoafFirstNames());

**Figure 6.4.** Creating and writing out a FOAF profile in Elmo.

As we see in this example, after creating the repository all the interaction with

the contents of the repository is encapsulated by the *ElmoManager* class, which is

used to load and instantiate the JavaBean. After setting some of the properties of the

*Person* instance, we write it out as an RDF/XML document.

An additional module of the Elmo API, the AugurRepository, can be used to

improve the performance of applications through (predictive) caching. Information

read from the repository is cached for further queries. (Similarly, writes are also

cached until the transaction is committed. The default, however, is automatic commit.)

Caching also involves predicting the kind of queries the user is likely to ask and

pre-loading the information accordingly. Already when a resource is first accessed

all the properties of that resource are preloaded. Another strategy requires keeping

6.1 Building Semantic Web applications with social network features 131

track of the queries from which resources have been retrieved. If later a property is

read on such a resource, the same property is retrieved for all the resources originating

from the same query. Optionally, the query re-writing introduced in Section 5.4.4

can also be performed by Elmo: for example, when executing the *getName* method

on a Person instance not only the names of current instance is returned, but also all

the names of all instances that are *owl*:*sameAs* the current instance.

Lastly, Elmo helps developers to design applications that are robust against incorrect

data, which is a common problem when designing for the Web. In general,

Semantic Web applications processing external data typically have few guarantees

for the correctness of the input. In particular, many of the RDF documents on the

Web —especially documents written by hand—, are either syntactically incorrect,

semantically inconsistent or violate some of the assumptions about the usage of the

vocabularies involved. Most of these problems result from human error. For example,

many users of FOAF mistakenly assume that the value of the *foaf* :*mbox* property

should be a Literal. In reality, the ontology expects a URI that encodes the email address

using the mailto protocol, e.g. *mailto:pmika@cs.vu.nl*.

Syntax can be easily checked by syntax validators such as the online RDF validation

service of theW3C1112 Inconsistency can be checked by OWL DL reasoners.

13 Elmo provides solutions for performing checks that can only be carried out programmatically,

for example checking if the value of the *foaf* :*mbox* property begins

with the mailto: prefix (protocol identifier). (The mistake of using a Literal would

also be found by an OWL DL reasoner, because the *foaf* :*mbox* property is declared

to be an *owl*:*ObjectProperty*.)

Using aspect-oriented programming, interceptors can be added to setter methods

of JavaBeans in order to validate information that is being inserted to the repository.

On the other hand, *validators* can be written for checking existing data for correctness.

It is the choice of the programmer whether to stop processing when such check

fail, or rather try to recover, for example by removing or correcting erroneous data.

**Elmo tools**

Elmo also contains a number of tools to work with RDF data. The Elmo *scutter*

is a generic RDF crawler that follows *rdfs*:*seeAlso* links in RDF documents, which

typically point to other relevant RDF sources on the web.14 RDF(S) seeAlso links

are also the mechanism used to connect FOAF profiles and thus (given a starting

location) the scutter allows to collect FOAF profiles from the Web.

Several advanced features are provided to support this scenario:

11 http://www.w3.org/RDF/Validator/

12 Another possible strategy to deal with invalid data would be to create “forgiving parsers”

that correct common user mistakes on the fly and without requiring the involvement of the

user. This is the way web browsers work when tolerating many of the mistakes authors

make when creating HTML pages.

13 Recall that RDF ontologies can not be inconsistent, except for the rare case of datatype

inconsistency.

14 The Elmo scutter is based on original code by Matt Biddulph for Jena.

132 6 Developing social-semantic applications

*•* Blacklisting: sites that produce FOAF profiles in large quantities are automatically

placed on a blacklist. This is to avoid collecting large amounts of uninteresting

FOAF data produced by social networking and blogging services or other

dynamic sources.

*•* Whitelisting: the crawler can be limited to a domain (defined by a URL pattern).

*•* Metadata: the crawler can optionally store metadata about the collected statements.

This metadata currently includes provenance (what URL was the information

coming from) and timestamp (time of collection)

*•* Filtering: incoming statements can be filtered individually. This is useful to remove

unnecessary information, such as statements from unknown namespaces.

*•* Persistence: when the scutter is stopped, it saves its state to the disk. This allows

to continue scuttering from the point where it left off. Also, when starting the

scutter it tries to load back the list of visited URLs from the repository (this

requires the saving of metadata to be turned on).

*•* Preloading from Google: the scutter queue can be preloaded by searching for

FOAF files using Google

*•* Logging: The Scutter uses Simple Logging Facade for Java (SLF4J) to provide a

detailed logging of the crawler.

The task of the Elmo *smusher* is to find equivalent instances in large sets of data,

which is the problem we discussed in Section 5.4. This is a particularly common

problem when processing collections of FOAF profiles as several sources on the

Web may describe the same individual using different identifiers or blank nodes.

Elmo provides two kinds of smushers that implement strategies to smushing.

The first kind of smusher uses class-specific comparators for comparing instances.

Implementations are given for comparing *foaf* :*Person* objects based on name, email

addresses and other identifying properties. There is also a comparator for comparing

publications based on a combination of properties.

The second kind of smusher compares instances in a repository based on a certain

property, i.e. in this case smushing proceeds property-by-property instead of

instance-by-instance. For example, inferring equality based on inverse functional

properties can be done with a single query for all such properties:

CONSTRUCT {x} owl:sameAs {y} FROM

{prop} rdf:type {owl:InverseFunctionalProperty},

{x} prop {v}, {y} prop {v}

USING NAMESPACE

foaf = <http://xmlns.com/foaf/0.1/>,

example = <http://www.example.org/>,

owl = <http://www.w3.org/2002/07/owl#>

When resolving such a CONSTRUCT query first the graph pattern described

after the FROM keyword is matched against the repository and for every occurrence

6.1 Building Semantic Web applications with social network features 133

the variables are bound to actual values.15 With these bindings a set of new graphs

is constructed by filling the variables in the pattern described in front of the FROM

keyword. These graphs are merged and returned in a single RDF document. Notice

that the query will also infer *owl*:*sameAs* relations where *x* = *y*, although only for

instances that do have at least one value specified for at least one inverse functional

property. This can be prevented by adding an additionalWHERE clause.

The smushers report the results (the matching instances) by calling methods on

registered listeners. We provide several implementations of the listener interface,

for example to write out the results in HTML, or to represent matches using the

*owl*:*sameAs* relationship and upload such statements to a Sesame repository.

Smushers can also be used as a *wrapper*. The difference between a wrapper and

a smusher is that a smusher finds equivalent instances in a single repository, while a

wrapper compares instances in a source repository to instances in a target repository.

If a match is found, the results are lifted (copied) from the source repository to the

target repository. This component is typically useful when importing information

into a specific repository about a certain set of instances from a much larger, general

store.

**6.1.4 GraphUtil**

GraphUtil is a simple utility that facilitates reading FOAF data into the graph object

model of the Java Universal Network Graph (JUNG) API. GraphUtil can be configured

by providing two different queries that define the nodes and edges in the RDF

data. These queries thus specify how to read a graph from the data. For FOAF data,

the first query is typically one that returns the *foaf* :*Person* instances in the repository,

while the second one returns *foaf* :*knows* relations between them. However, any other

graph structure that can be defined through queries (views on the data) can be read

into a graph.

JUNG16 is a Java library (API) that provides an object-oriented representation of

different types of graphs (sparse, dense, directed, undirected, k-partite etc.) JUNG

also contains implementations for the most well known graph algorithms such as

Dijkstra’s shortest path. Various implementations of the *Ranker* interface allow to

compute various social network measures such as the different variations of centrality

described in Section 2.3.3. We extended this framework with a new type of

ranker called *PermanentNodeRanker* which makes it possible to store and retrieve

node rankings in an RDF store.

15 From a practical perspective, it is also worth noting that the order of the graph expressions

in the query does matter with respect to performance. Queries are evaluated from left to

right by most engines and there it is reasonable to put in from the pattern that produces the

least matches. In our case the first triple pattern contains only one variable (property) and

that can only be bound to a small number of values (the number of inverse functional properties).

The other two triple patterns contain three variables and thus match all statements

in the repository. Putting them in front would result in a very inefficient query resolution.

16 http://jung.sourceforge.net

134 6 Developing social-semantic applications

Lastly, JUNG provides a customizable visualization framework for displaying

graphs.Most importantly, the framework let’s the developer choose the kind of layout

algorithm to be used and allows for defining interaction with the graph visualization

(clicking nodes and edges, drag-and-drop etc.) The visualization component can be

used also in applets as is the case