Department of Computer Science and Engineering

Subject Name:Semantic web and social networks

Year and Semester:III-I

Regulations:R18

Syllabus

MODULE I: **World Wide Web**

Web Intelligence - Thinking and Intelligent Web Applications, The Information Age, The World Wide Web, Limitations of today's Web, The Next Generation Web, Machine Intelligence, Artificial Intelligence.

Web Description - Ontology, Inference Engines, Software Agents, Berners-Lee www, Semantic Road Map, Logic on the semantic Web.

MODULE II:Knowledge Representation for the Semantic Web

Ontology - Ontologies and their role in the semantic web, Ontologies Languages for the Semantic Web -Resource Description Framework (RDF) / RDF Schema.

Web Languages - Ontology Web Language (OWL), UML, XML, XML Schema.

MODULE III: **Ontology Engineering**

Ontology Development - Ontology Engineering, constructing Ontology, Ontology Development Tools, Ontology

Ontology Sharing and Merging - Ontology Sharing and Merging, Ontology Libraries and Ontology mapping, Logic, Rule and Inference Engines.

MODULE IV: Semantic Web Applications, Services and Technology

Semantic Web Services - Semantic Web applications and services, Semantic Search, e-learning

Semantic Web Applications - Semantic Bioinformatics, Knowledge Base, XML Based Web Services, Creating an OWL-S Ontology for Web Services, Semantic Search Technology, Web Search Agents and Semantic Methods

MODULE V:Social Network Analysis and Semantic Web

Social Network Analysis - What is social Networks analysis, development of the social networks analysis, Electronic Sources for Network Analysis - Electronic Discussion networks.

Semantic Web - Blogs and Online Communities, Web Based Networks, Building Semantic Web Applications with social network features.

TEXT BOOKS: 1. Berners Lee, Gödel and Turing, "Thinking on the web", Wiley interscience, 2008. 150 2.Peter Mika, ".Social Networks and the Semantic Web", Springer, 2007.

REFERENCES: 1. J.Davies, R.Studer, P.Warren, Johri. Wiley & Sons, "Semantic Web Technologies, Trends and Research in Ontology Based Systems" 2. Liyang Lu Chapman and Hall, " Semantic Web and Semantic Web Services",

MODULE I

World Wide Web

Web Intelligence: Today, the world is experiencing the excitement of an historic change. We find ourselves in the midst of an information revolution, the result of rapid advances in technology built in great part upon the shoulders of three pivotal pioneers: Kurt Gödel, Alan Turing, and Tim Berners-Lee. Through their contributions, we are witnessing the remarkable refashioning of the Information Age, which began in the 1950s, into the Information Revolution as the World Wide Web evolves into a resource with intelligent capabilities.

Web intelligence is an issue of philosophy as much as application. It has beensuggested that the next generation of Web architecture, the Semantic Web, createsan Artificial Intelligence (AI) application that will make Web content meaningfulto computers thereby unleashing a revolution of new abilities. More realistically,however, the Semantic Web will add semantics to the Web along with somelimited AI capabilities to produce a more useful Web. The balance betweengreater logic expressive power and solvable computer reasoning complexity isstill being questioned and evaluated.

Alan Turing was one of the great thinkers of the twentieth century, and hiscontributions in the area of machine intelligence were seminal. This chapterprovides an overview of Turing's contributions and discusses some of the keyideas emanating from his work. In addition, we engage in a discussion of themeaning of machine intelligence and offer some perspective on how makingcontent on the Web machine processible will contribute toward Web intelligence.

THINKING AND INTELLIGENT WEB APPLICATIONS

When the philosopher Rene Descartes proclaimed his famous observation "Cogito,ergo sum," he demonstrated the power of thought at the most basic level by derivingan important fact (i.e., the reality of his own existence) from the act of thinkingand self-awareness.

Today, the term "thinking" is frequently loosely defined and ambiguouslyapplied. For that reason, it is important to provide a brief preview of whatwe mean by the term in the context of intelligent applications on the WorldWide Web.

In general, thinking can be a complex process that uses concepts, their interrelationships,and inference or deduction, to produce new knowledge. However,thinking is often used to describe such disparate acts as memory recall, arithmeticcalculations, creating stories, decision making, puzzle solving, and so on.

Some aspects of the concept of thinking can be inferred by recognizing thatan individual can be identified as intelligent if they have accurate memory recall,the ability to apply valid and correct logic, and the capability to expand theirknowledge through learning and deduction. Ultimately, self-awareness and consciousnessare important if not central aspects of human intelligence, but thesecharacteristics prove much more difficult to analyze or emulate than other, moredirect indicators of intelligence.

The term "intelligence" can be applied to nonhuman entities as we do in thefield of Artificial Intelligence (AI). But frequently we mean something somewhatdifferent than in the case of human intelligence. For example, while one mightbe quite impressed with the intelligence of a child prodigy who can performdifficult arithmetic calculations quickly and accurately, a computer that couldperform the same calculations faster and with greater accuracy would not beconsidered to be particularly intelligent. An individual who has rapid memoryrecall and who has accumulated sufficient amounts of information to consistentlywin games such as Scrabble, or Trivial Pursuit, might also be considered to bevery intelligent; while a computer storing much greater quantities of accessiblefactual information would not.

It is recognized that human thinking involves complicated interactions withinthe biological components of the brain, and that the process of learning is alsoan important element of human intelligence. Increasingly, software applicationsperform tasks that are sufficiently complex and human-like that the term intelligentmay be appropriate. Whereas AI can be seen as the science of machines that behave intelligently (or simulate intelligent behavior), the concept of intelligentapplications entails the efforts to take advantage of AI technologies to enhanceapplications and make them act in more intelligent ways.

THE INFORMATION AGE

We are accustomed to living in a world that is rapidly changing. This is truein all aspects of our society and culture, but is especially true in the field ofinformation technology. Most are aware of the rapid advances in computer andinformation technology as exemplified in "Moore's law," the observation madein 1965 by Gordon Moore, co-founder of Intel that the number of componentson integrated circuits had doubled every 18 months.

As a result, it is common to observe such rapid change and comment simplythat "things change." But, even accepting the reality of rapid change, when canwe assess that the change has actually improved human productivity? And whattypes of change can produce transformation on a global scale?

To gain an historical perspective of global change, take a brief look back.Over the millennia, mankind has experienced two global revolutionary changes: the Agricultural Revolution and the Industrial Revolution. Each produced overa 100-fold factor of improvement in the access to basic human resources andsubsequently freed individuals to pursue higher level cultural and social goals.In addition, over the past half century, many have been pondering the possibilitythat the technological inventions of the Information Age may in fact be of suchscope as to represent a third revolutionary change: the Information Revolution.

Should the rapidly changing world of the Information Age be considered aglobal revolutionary change on the scale of these earlier revolutions? In orderto address this issue we must compare it with the changes associated with theAgricultural Revolution, which began around 8000 B.C. and continued through around 1700 A.D., and the Industrial Revolution, which began around 1700 andis still continuing to spread across the underdeveloped world even today.

Ten thousand years ago, humans lived in migratory groups and with theaid of flexible, rapidly evolving cultures, these loosely organized groups of"hunter–gatherers" were able to adapt to virtually all the climate zones and environmentalniches on the planet, from the Arctic to temperate zones to the tropics.They fed themselves by hunting, herding, fishing, and foraging. The essence ofhunting and gathering economies was to exploit many resources lightly ratherthan to depend heavily on only a few. Small, mobile human populations

subsistedon whatever resources were available within their territory. In such small,continuously moving communities, there was little opportunity for economic orother kinds of specialization to develop. What one person knew and believed,the entire group tended to know and believe. Life was communal; cultural andtechnical knowledge and skills were widely diffused.

However, a major and dramatic turning point in human social developmentoccurred when humans discovered the utility of agriculture. Agriculture resultedin living permanently in one place. Living in one spot permanently means exploitinga relatively small amount of land very intensively and over a long periodof time.

To survive, agriculturalists had to collect all their food for the year at one ortwo harvest times, rather than gathering year round. Nothing, therefore, couldbe allowed to interrupt the harvest. This is due to a very narrow window ofopportunity for planting and cultivating. Under this kind of pressure, agriculturalcommunities became more time-conscious. Agriculturalists also had to store theproduce of their fields for the rest of the year, protect it from moisture, vermin,and thieves, and learn to distribute supplies so the community could survive andstill have seed for next year's planting. These conditions created a new kind oflife style.

While a hunter–gather acquired resources from 100 acres to produce an adequatefood supply, a single farmer needed only 1 acre of land to produce theequivalent amount of food. It was this 100-fold improvement in land managementthat fueled the agricultural revolution. It not only enabled far more efficient foodproduction, but also provided food resources well above the needs of subsistence,resulting in a new era built on trade.

The Agricultural Revolution crept slowly across villages and regions, introducingland cultivation and a new way of life. During the long millennia thatthis revolution progressed, the world population was divided into two competitivecategories: primitive and civilized. The primitive tribes continued in themode of hunting–gathering while the civilized communities worked the land.The civilized communities produced foodstuffs for their own use with a surplusto allow for trade.

Because farmers consumed what they produced directly and traded their surpluslocally, there was a close relationship between production and consumption.However, as trade developed the Agricultural Revolution encouraged the construction of the roads that facilitated the exchange of specialized produce onan expanding scale until it eventually become global.

This evolutionary transition to an agricultural basis for society was still incompletewhen, by the end of the seventeenth century, the Industrial Revolutionunleashed a new global revolutionary force. Societies, up until this period, hadused human and animal muscle to provide the energy necessary to run theeconomy. As late as the French revolution, millions of horses and oxen providedthe physical force that supported the European economy.

Where a single farmer and his horse had worked a farm, during the IndustrialRevolution, workers were able to use a single steam engine that produced 100times the horsepower. Consequently, the Industrial Revolution placed a 100-foldincrease of mechanical power into the hands of the laborer. It resulted in thefalling cost of labor and this fueled the economic growth of the period. Thenew industrialization process moved rapidly over Europe and across the othercontinents. It utilized flowing water, wood, coal, oil, and gas to generate energythat in turn produced an abundance of food and material goods.

In contrast to the agricultural cycle of planting and harvesting, the industrialsociety followed the continuous linear timing of machines to build inventoryand maintain stored goods. This enabled consumers to be far removed from theproducer. The industrialization process, therefore, broke down the close relationshipbetween local production and consumption. The result was a stockpiling ofresources at strategic locations along the distribution path. Again this revolutionarychange also stimulated the intellectual growth of the society in order to meetthe skill requirements for the workers.

The Industrial Revolution was defined by the application of power-drivenmachinery to manufacturing. It was not until 1873 that a dynamo capable ofprolonged operation was developed. Through the nineteenth century the use ofelectric power was limited by small productive capacity, short transmission lines,and high cost. The coming of the railroads greatly facilitated the industrializationprocess and the building of transcontinental railroads mimicked the early growthof roads during the beginning of the Agricultural Revolution.

The Industrial Revolution became characterized by six basic characteristics:Standardization: mass production of identical parts. Concentration: work andenergy maintained locally. Centralization: authoritative leadership. Specialization:division of labor. Synchronization: work at the pace of machines. Maximization:strategic planning.

One important development was the construction of the railroads that facilitatedthe exchange of raw materials into finished products on a global scale.

The 1950s—the decade that introduced the computer—began the latest historicturning point, the Information Age. However, it did not approach its full potentialtoward reducing information transaction costs until the computer was networkedfor global communications beginning in the 1990s with the growth of the Internet.

THE WORLD WIDE WEB

How is the World Wide Web managing knowledge and empowering the InformationRevolution? Does rapid change and improved information productivityrequire more intelligent Web capabilities? What technologies offer the best opportunitiesfor sustained powerful change? Let us explore these questions by brieflyevaluating the development and limitations of today's Web technology.

The history of the Web extends back more than 40 years. Looking back, wecan find early signs of network architecture in the 1960s. The RAND Corporationbegan research sponsored by the U.S. Air Force to determine how to developrobust, distributed communication networks for military command and controlthat could survive and function in the event of a nuclear attack.

This initial study led to the development of the Advanced Research ProgramsAgency Network (ARPANET) an agency of the U. S. Department of Defense. Inaddition to robustness, it promoted the sharing of supercomputers among scientificresearchers in the United States. ARPANET originally consisted of four nodes inthe western U.S. (the University of California at Los Angeles, SRI of Stanford,California, the University of California at Santa Barbara, and the University ofUtah) connected in a system that was to become the precursor to the Internet.

The ARPANET was a success right from the very beginning. Over thosefirst few years, the network developed and expanded as new sites (nodes) wereadded, and as new capabilities and features were introduced, such as softwareand protocols to facilitate email and file transfers. Although the ARPANET wasoriginally designed to allow scientists to share data and access remote computers,email quickly became the most popular application.

The ARPANET became ahigh-speed digital post office as people used it to collaborate on research projects.It was a distributed system of "many-to-many" connections.

Transmission Control Protocol/Internet Protocol (TCP/IP), a suite of networkcommunications protocols used to connect hosts on the Internet was developedto connect separate networks into a "network of networks" (e.g., the Internet).These protocols specified the framework for a few basic services that everyonewould need (file transfer, electronic mail, and remote logon) across a very largenumber of client and server systems. Several computers linked in a local networkcan use TCP/IP (along with other protocols) within the local network just asthey can use the protocols to provide services throughout the Internet.

The IPcomponent provides routing from the local to the enterprise network, then toregional networks, and finally to the global Internet. Socket is the name for thepackage of subroutines that provide access to TCP/IP.

The mid-1980s marked a boom in the personal computer and superminicomputerindustries. The combination of inexpensive desktop machines and powerful,network-ready servers allowed many companies to join the Internet for thefirst time.

Corporations began to use the Internet to communicate with each other andwith their customers. By 1990, the ARPANET was decommissioned, leaving onlythe vast network-of-networks called the Internet with over 300,000 hosts.

The stage was set for the final step to move beyond the Internet, as threemajor events and forces converged, accelerating the development of informationtechnology. These three events were the introduction of the World WideWeb, the widespread availability of the graphical browser, and the unleashing ofcommercialization.

In startling contrast, AOL, CompuServe, and Microsoft were investing fortunesin proprietary networks that offered mostly duplicated and limited amounts ofinformation to the public, but for a fee. Tim Berners-Lee on the other hand wasdesigning a cheap, efficient, and simple way for universal access to great storesof information for free.

In 1991, Berners-Lee, working at European Particle Physics Laboratory ofthe European Organization for Nuclear Research, Conseil European pour laRecherché Nucl´eaire, (CERN) in Switzerland, introduced the concept of theWorld Wide Web.

The Web combined words, pictures, and sounds on Internet pages and programmerssaw the potential for publishing information in a way that could be aseasy as using a word processor, but with the richness of multimedia.

Berners-Lee and his collaborators laid the groundwork for the open standardsof the Web. Their efforts included the Hypertext Transfer Protocol (HTTP) linkingWeb documents, the Hypertext Markup Language (HTML) for formatting Webdocuments, and the Universal Resource Locator (URL) system for addressingWeb documents.

Today, we reach the Web through commercial browsers, such as, InternetExplorer or Netscape Navigator. These browsers are powerful applications thatread the markup languages of the Web display their contents and collect data.

The primary language for formatting Web pages is HTML. With HTML theauthor describes what a page should look like, what types of fonts to use, whatcolor the text should be, where paragraph marks come, and many more aspectsof the document. All HTML documents are created by using tags. Tags havebeginning and ending identifiers to communicate to the browser the beginningand ending text formatted by the tag in question.

In 1993, Marc Andreessen and a group of student programmers at NCSA(the National Center for Supercomputing Applications located on the campusof University of Illinois at Urbana Champaign) developed a graphical browserfor the World Wide Web called Mosaic, which he later reinvented commerciallyas Netscape Navigator. The graphical browser greatly stimulated Webdevelopment.

Soon studies of Web traffic began to show signs that all Web sites were not"equidistant." That is, some sites were acting as hubs and garnishing a dominantshare of the "through" traffic. In addition, some Web sites acted as prominentsources of primary content, and became authorities on the topic, while othersites, resembled high-quality guides acted as focused hub, directing users torecommended sites. By 1994, the W3C was founded under the leadership of TimBerners-Lee to develop standards for the Web.

LIMITATIONS OF TODAY'S WEB

Over the past several decades, the Web has changed from a distributed, high reliability,open system, to a Web dominated by portals, such as Yahoo, Google, AOL, and MSN, which control much of the traffic. While the W3C developedopen Web standards, vendors have been customizing their applications for efficientbusiness logic processing through their proprietary servers and applications.

By the year 2000, the introduction of Web Services led to a dichotomy ofMicrosoft's Windows (.NET) and Sun's Java (J2EE) frameworks within theserver community infrastructure. As a result, the Web moved strongly towardbecoming a decentralized network with highly critical hubs. The extensibleMarkup Language (XML) was developed as a markup language based onthe principles and rules of Standard Generalized Markup Language (SGML)and uses tags that are not predefined. This gives XML great flexibility, andextensibility. The XML remains the interoperable bridge for exchanging databetween J2EE and .NET, and as a result XML is an essential support for bothWeb Services' frameworks.

Nevertheless, the problem with performing intelligent tasks, such as automatedWeb Services, is that they first require human assistance, and secondthat they must rely on the interoperation and inefficient exchange of the twocompeting proprietary server frameworks to successfully communicate complexbusiness logic.

The Web is still based on HTML, which describes how information is to bedisplayed and laid out on a Web page for humans to read. In effect, the Webhas developed as a medium for display of information directly to humans; therehas been no real emphasis on establishing the capability for machine understandingand processing of web-based information. HTML is not capable of beingdirectly exploited by information retrieval techniques; hence processing of Webinformation is largely restricted to manual keywords searches.

Because the World Wide Web has its origin in the idea of hypertext, theWeb is focused on textual data enriched by illustrative insertions of audiovisualmaterials. The status quo paradigm of the Web is centered on the client–serverinteraction, which is a fundamentally asymmetric relationship between providersinserting content onto the Web hypertext (the server) and users who essentiallyread texts or provide answers to questions by filling out forms (the clients).

Today, the development complex networks of meaningful content remainsdifficult. Web browsers are restricted to accessing existing information in a standard form. In addition, some of today's basic Web limitations include search,database support, interoperable

applications, intelligent business logic, automation,security, and trust. As a result, the Information Revolution awaits the nextbreak-through to fully open the information flow.

THE NEXT GENERATION WEB

A new Web architecture called the Semantic Web offers users the ability to workon shared knowledge by constructing new meaningful representations on theWeb. Semantic Web research has developed from the traditions of AI and ontologylanguages and offers automated processing through machine-understandablemetadata.

Semantic Web agents could utilize metadata, ontology's, and logic to carry outits tasks. Agents are pieces of software that work autonomously and proactivelyon the Web to perform certain tasks. In most cases, agents will simply collect andorganize information. Agents on the Semantic Web will receive some tasks to performand seek information from Web resources, while communicating with otherWeb agents, in order to fulfill its task.

MACHINE INTELLIGENCE

Machine intelligence is associated with Machine Learning, Computational Intelligence,Soft-Computing, and Artificial Intelligence. Although these terms areoften used interchangeably, they are actually different branches of study.

For example, Artificial Intelligence involves symbolic computation (i.e.,the mathematical transformation of symbolic expressions, using computeralgorithms), while Soft-Computing (i.e., techniques in computer scienceand in artificial intelligence, such as Fuzzy logic, Neural networks, andProbabilistic reasoning, that resemble human reasoning more closely thantraditional techniques) involves intensive numerical computation. The followingsub branches of machine intelligence have particular relevance to the SemanticWeb and the idea Web intelligence: computational complexity, descriptive logic,ontology, inference, and software agents.

Although symbolic AI is currently built and incorporated into Semantic Webdata representation, there is no doubt that software tool developers will eventuallyincorporate the soft-computing paradigm as well. The benefit of such a stepwill be the creation of adaptive software. This would imply that soft-computingapplications will have the ability to adapt to changing environments and input.

While the Semantic Web is under development, concepts surrounding machineintelligence will continue to evolve. The extent of the usefulness of the SemanticWeb will be tested in various ways, but the controversy involving the meaningof machine intelligence will undoubtedly not end in the foreseeable future.

In the following sections, topics related to semantic networks and descriptionlogic, will be discussed and then each of the key machine-intelligence areas identifiedabove will be addressed starting with computational complexity, followedby knowledge representation, Ontology, inference engines, and software agents.

Alan Turing, while acting as the leading cryptography expert to break the Germancodes during the Second World War, formulated the ideas that emerged after the war as Intelligent Machinery, and are now referred to as AI. Key to this field ofstudy is the definition of what is meant by the terms "thinking" and "intelligence."

Thinking is often ambiguously defined, but generally can be applied to acomplex processes that uses concepts, their interrelationships, and inference toproduce new knowledge. We can extend the concept of thinking and identify anintelligent individual as one who is capable of accurate memory recall or able toapply logic to extend knowledge.

It is possible to extend the description of intelligence to nonhuman entities aswell, such as in AI. But we frequently mean something different than in the caseof human intelligence. For example, while one might be quite impressed with theintelligence of a child prodigy who can perform difficult arithmetic calculationsquickly and accurately, a computer that could perform the same calculationsfaster and with greater accuracy would not be considered intelligent.

While it is still not possible to resolve controversial differences of opinion overthe nature of human intelligence, it is possible to recognize certain attributes thatmost would agree reflect the concept. These include such elements as: the abilityto learn; the ability to assimilate; the ability to organize and process information;and the ability to apply knowledge to solve complex problems. By extension then,many of these attributes of human intelligence can be traced into the various areasof research in the field of artificial intelligence. Artificial intelligence addressesthe basic questions of what it means for a machine to have intelligence.

In 1947, shortly after the end of World War II, English mathematician AlanTuring first started to seriously explore the idea of intelligent machines. By1956, John McCarthy of MIT coined the term Artificial Intelligence, and bythe late 1950s, there were many researchers in AI, most basing their work onprogramming computers. Eventually, AI became more than a branch of science:it expanded far beyond mathematics and computer science into fields such asphilosophy, psychology, and biology.

ARTIFICIAL INTELLIGENCE

How far is AI from reaching human-level intelligence? Some have suggested thathuman-level intelligence can be achieved by writing large numbers of programsand assembling vast databases of facts in the languages used for expressingknowledge. However, most AI researchers believe that new fundamental ideasare required before true intelligence can be approached.

There are two main lines of AI research. One is biological, based on theidea that since humans are intelligent, AI should study humans and imitate thepsychology or physiology of human intelligence. The other is phenomenological,based on studying and formalizing common sense facts about the world and theproblems that the world presents to the achievement of goals thereby providingfunctionality that, in humans, would be considered intelligent behavior, even ifthe approach used is quite different from what would be found in a human.

Today, AI still means different things to different people. Some confusionarises because the word intelligence is so ill-defined. Artificial intelligence issometimes described in two ways: strong AI and weak AI. Strong AI asserts thatcomputers can be made to think on a level (at least) equal to humans. Weak AIsimply holds that some thinking-like features can be added to computers to makethem more useful tools. Examples of Weak AI abound: expert systems, drive by-wire cars, smart browsers, and speech recognition software. These weak AIcomponents may, when combined, begin to approach some aspects of strong AI.

As participants of the Information Revolution, we could ask by extension,"What is Web intelligence?" For the most part, the World Wide Web can be consideredto be a massive information system with interconnected databases andremote applications providing various services. While these services are becomingmore and more user oriented, the concept of smart or intelligent applicationsand services on the Web is still in its infancy.

A classic example of an AI application that many would consider intelligent insome form is computer chess. Over the years, computer chess-playing softwarehas received

considerable attention, and such programs are a commercial successfor home PCs or on the Web. In addition, most are aware of the highly visiblecontest between IBMs Deep Blue Supercomputer and the reigning World ChessChampion, Garry Kasparov in May 1997.

Millions of chess and computing fansobserved this event in real-time where, in a dramatic sixth game victory, DeepBlue beat Kasparov. This was the first time a computer has won a match with acurrent world champion under tournament conditions.

Computer chess programs generally make use of standardized openingsequences, and end game databases as a knowledge base to simplify these phasesof the game. For the middle game, they examine large trees and perform deepsearches with pruning to eliminate branches that are evaluated as clearly inferiorand to select the most highly favourable move.

Web Description:

ONTOLOGY

If a program wants to compare conceptual information across two knowledgebases on the Web, it has to know when any two given terms are being used tomean the same thing. Ideally, the program must have a way to discover commonmeanings for whatever knowledge bases it encounters. A solution to this problemis provided by the Semantic Web in the form of collections of information calledontologies. Artificial Intelligence and Web researchers use the term ontology asa document that defines the relations among terms. A typical ontology for theWeb uses taxonomy and a set of inference rules.

The taxonomy defines classes of objects and relations among them. Classes,subclasses, and relations among entities are important tools. We can express alarge number of relations among entities by assigning properties to classes andallowing subclasses to inherit such properties.

Inference rules in ontologies may express rules for manipulating information.Inference rules may express the rule: "If a city code is associated with a statecode, and an address uses that city code, then that address has the associatedstate code." A program could then readily deduce, for example, that a CornellUniversity address, being in Ithaca, must be in New York State, which is in theUnited States, and therefore should be formatted to U.S. standards.

The real power of the Semantic Web will be realized when people create manyprograms that collect Web content from diverse sources, automatically processthe information, and exchange the results. The effectiveness of software agentswill increase exponentially as more machine-readable Web content and automatedservices become available. The Semantic Web promotes this synergy: even agentsthat were not expressly designed to work together can transfer semantic data.

INFERENCE ENGINES

Inference engines are intended to derive answers from a knowledge base. Theyare at the core of the expert systems that provide a methodology for reasoningabout the information in the knowledge base, and for formulating conclusions.Inference engines process knowledge available in the Semantic Web. They deducenew knowledge from previously established knowledge.

An inference engine controls overall execution of a set of rules. It searchesthrough a knowledge base, attempting to pattern-match facts or knowledge presentin memory to the antecedents of rules. If a rule's antecedent is satisfied, the ruleis ready to fire and is placed in

the agenda. When a rule is ready to fire it meansthat since the antecedent is satisfied, the consequent can be executed.Salience is a mechanism used by some expert systems to add a proceduralaspect to rule inferencing. Certain rules may be given a higher salience thanothers, which means that when the inference engine is searching for rules to fire,it places those with a higher salience at the top of the agenda.

SOFTWARE AGENTS

An intelligent agent is a computer system that is situated in some environment,and that is capable of autonomous action and learning in order to meet its designobjectives. Agents have the following characteristics: they are reactive—theyperceive their environment, and respond; proactive—they exhibit goal-directedbehavior; and social—they interact with other agents.

Real-time intelligent agent technology offers a powerful Web tool. Agents areable to act without the intervention of humans or other systems; they have controlboth over their own internal state and their behavior. Normally, an agent willhave a repertoire of actions available to it. So that in complexity domains, agentsmust be prepared for the possibility of failure. This situation is called nondeterministic.Its set of possible actions represents the agent's capability to modifyits environments. Similarly, the action "purchase a house" will fail if insufficientfunds area available to do so. Actions therefore have preconditions associatedwith them, which define the possible situations in which they can be applied.

The key problem facing an agent is that of deciding which of its actions itshould perform to satisfy its design objectives. Agent architectures are reallysoftware architectures for decision-making systems that are embedded in anenvironment. The complexity of the decision-making process can be affectedby a number of different environmental choices: accessible versus inaccessible,deterministic versus nondeterministic, episodic versus nonepisodic, static versusdynamic, and discrete versus continuous.

The most complex general class of environments consists of those that areinaccessible, nondeterministic, no episodic, dynamic, and continuous. For theSemantic Web, providing sufficient expressive power for agents to interact successfullyis essential.

BERNERS-LEE: WHAT IS SOLVABLEON THE WEB?

When Tim Berners-Lee was developing the key elements of the World Wide Web,he showed great insight in providing Hypertext Markup Language (HTML) as asimple easy-to-use Web development language. As a result, it was rapidly andwidely adopted. To produce Web information required skills that could be learnedwith a high school level education. Consequently, personal computing mergedwith global networking to produce the World Wide Web.

The continuing evolution of the Web into a resource with intelligent features,however, presents many new challenges. The solution of the World Wide WebConsortium (W3C) is to provide a new Web architecture that uses additional layersof markup languages that can directly apply logic. However, the addition ofontology's, logic, and rule systems for markup languages means consideration ofextremely difficult mathematic and logic consequences, such as paradox, recursion,undesirability, and computational complexity on a global scale. Therefore,it is important to find the correct balance between achieving powerful reasoningwith reasonable complexity on the Web. This balance will decide what is solvableon the Web in terms of application logic.

THE WORLD WIDE WEB

By 1991, three major events converged to accelerate the development of the InformationRevolution. These three events were the introduction of the World WideWeb, the widespread availability of the graphical browser, and the unleashing ofcommercialization on the Internet. The essential power of the World Wide Webturned out to be its universality though the use of HTML. The concept providedthe ability to combine words, pictures, and sounds (i.e., to provide multimediacontent) on Internet pages. This excited many computer programmers who sawthe potential for publishing information on the Internet with the ease of using aword processor, but with the richness of multimedia forms.

Berners-Lee and his collaborators laid the groundwork for the open standardsof the Web. Their efforts included inventing and refining the Hypertext TransferProtocol (HTTP) for linking Web documents, the HTML for formatting Webdocuments and the Universal Resource Locator (URL) system for addressingWeb documents.

Hypertext Markup Language is the primary language for formatting Webpages. With HTML, the author describes what a page should look like, whattypes of fonts to use, what color text should be, where paragraphs begin, andmany other attributes of the document.**Hypertext Transfer Protocol**

Hypertext Transfer Protocol is the network protocol used to deliver files anddata on the Web including: HTML files, image files, query results, or anythingelse. Usually, HTTP takes place through TCP/IP sockets. Socket is the term forthe package of subroutines that provide an access path for data transfer throughthe network.

Like most network protocols, HTTP uses the client–server model: An HTTPclient opens a connection and sends a request message to an HTTP server; theserver then returns a response message, usually containing the resource that wasrequested. After delivering the response, the server closes the connection.

A simple HTTP exchange to retrieve the file at the URL, first opens a socketto the host www.somehost.com, at port 80 (the default) and then, sends followingthrough the socket:

```
GET/path/file.html HTTP/1.0
From: someuser@somehost.com
User-Agent: HTTPTool/1.0
```

The server responds with the HTTP protocol file followed by the HTML "hello world" file with the following:

```
HTTP/1.0 200 OK
Date: Fri, 31 Dec 1999 23:59:59 GMT
Content-Type: text/html
Content-Length: 1354

<html>
<body>
Hello World
</body>
</html>
```

This simple process has proven to be not only easy and straightforward, buthighly successful as well.

Bridging Web Services

While the W3C continually develops open Web standards, vendors have beencustomizing their applications for efficient business logic processing throughtheir proprietary server applications. For example, Web Services communicatethrough open standards including Simple Object Access Protocol (SOAP) and Web Service Description Language (WSDL), but then the business logic is executedthrough server pages designed for specialized server frameworks (eitherJava 2 Platform Enterprise Edition (J2EE) or Microsoft .NET).

Simple Object Access Protocol (SOAP) is an implementation of XML thatrepresents one common set of rules about how data and commands are representedand extended. It consists of three parts: an envelope (a framework for describingwhat is in a message and how to process it), a set of encoding rules (for expressinginstances of application-defined data types), and a convention for representingremote procedure calls and responses. The SOAP messages are fundamentallyone-way transmissions from a sender to a receiver using HTTP binding. SimpleObject Access Protocol describes commands and parameters that can be passedbetween browsers and Web Services for both J2EE and .NET platforms.

Web Services Description Language (WSDL) describes networked XML-basedservices. It provides a way for service providers to describe the format of requeststo their systems regardless of the underlying protocol or encoding. It is a part ofthe effort of Universal Discovery and Description Identification (UDDI) initiativeto provide directories and descriptions of such on-line services for electronicbusiness.

Limits of Today's Web

The Web has changed from its original structure of a distributed, high-reliability,open system without a superimposed logic or metadata. Today, the basic informationis still displayed as a distributed open system, but the development ofportals, such as, Yahoo, Google AOL, and MSN has focused Web entry and ledto controlling traffic to partisan sites. In addition, business logic has migratedprimarily into two segregated server frameworks: active server pages and java
server pages. The result has produced a decentralized Web system with criticalproprietary portal-centric nodes and frameworks.

In the future, we can expect significant improvements, such as increased averagebandwidth, the use of open standards to facilitate advanced markup languages,the application of metadata, and the use of inference search.

The paradigm of the Web is centered on the client–server interaction, which isa fundamentally asymmetric relationship between providers, who insert contentinto the Web hypertext (server) and users who essentially read texts or provideanswers to questions by filling out forms (clients). The hyperlinks of the Webrepresent structures of meaning that transcend the meaning represented by individualtexts. At present, these Web structures of meaning lack longevity and canonly be blindly used, for example by search engines, which at best optimizenavigation by taking into account the statistical behavior of Web users.

In effect, the Web has developed as a medium for humans without a focus ondata that could be processed automatically. Hypertext Markup Language is notcapable of being

directly exploited by information retrieval techniques; hence theWeb is restricted to manual keyword searches.

The problem at present is that there is no way to construct complex networksof meaningful relations between Web contents. In fact, the providers have noinfluence on the links to the contents they provide and the users have no impacton the available access structures to the content. As a result, some of today'sbasic Web limitations include search, database support, interoperable applications,intelligent business logic, automation, security, and trust.An important framework for creating meaningful Web links can be providedby the Semantic Web: the automated creation of links between machine understandablemetadata. Such semantic linking will not be restricted to the useof specifically prepared metadata sets, but will exploit the meaningful structure ofthe Web itself in order to provide a content-based semantic access to information.

THE SEMANTIC WEB ROADMAP

The inventor of the World Wide Web, Tim Berners-Lee, and his W3C team workto develop, extends, and standardizes the Web's markup languages and tools. Inaddition, what they are designing is the next generation Web architecture: theSemantic Web.

Currently, the objective of the Semantic Web architecture is to provide aknowledge representation of linked data in order to allow machine processingon a global scale. This involves moving the Web from a repository of datawithout logic to a level where it is possible to express logic through knowledge representationsystems. The vision for the Semantic Web is to augment theexisting Web with resources more easily interpreted by programs and intelligent
Agents.

A defining application for the Semantic Web will be a more effective searchcapability. While today's search engines index HTML pages to find many answersand cover a huge part of the Web, they return many irrelevant pieces of information.There is no notion of "correctness" to such searches.

The difficulty of semantic search is perhaps its most important limitation.With the current state of the Web, there are two methods of gaining broaderinformation about documents. The first is to use a directory, or portal site, manuallyconstructed by searching the Web and then categorizing pages and links.The problem with this approach is that directories take a tremendous effort tomaintain. Finding new links, updating old ones, and maintaining the databasetechnology, all add to a portal's administrative burden and operating costs. Thesecond method uses automatic Web crawling and indexing systems.

Consequently, searching the World Wide Web can be frustrating. The resultof having better standard metadata could be a Web where users and agents coulddirectly tap the latent information in linked and related pages. This would be apowerful paradigm greatly improving the intelligent use of Web resources.

By contrast, logic engines have typically been able to restrict their output tothat which is a provably correct answer, but have suffered from the inability togo through the mass of connected data across the Web to construct valid answers.

If an engine of the future combines a reasoning engine with a search engine,it may actually be able to produce useful results. It will be able to reach out toindexes that contain very complete lists of all occurrences of a given term, andthen use logic to weed out all but those that can be of use in solving the givenproblem.

If the Semantic Web can bring structure and meaningful content to the Web,it will create an environment where software agents can carry out sophisticatedtasks for users. The first steps in weaving the Semantic Web with the existingWeb are already under way. In the near future, these developments will providenew functionality as machines become better able to "understand" and processthe data.

For the Semantic Web to function, computers must have access to structuredcollections of information and sets of inference rules that they can use to conductautomated reasoning. Artificial Intelligence researchers have long studiedsuch systems and have produced today's knowledge representation. Knowledgerepresentation is currently in a state comparable to that of hypertext before theadvent of the Web.

The objective of the Semantic Web therefore, is to provide a framework thatexpresses both data and rules for reasoning for Web-based knowledge representation.Adding logic to the Web means using rules to make inferences, choosecourses of action, and answering questions. A combination of mathematical andengineering issues complicates this task. The logic must be powerful enough todescribe complex properties of objects, but not so powerful that agents can betricked by being asked to consider a paradox.

The development of the Semantic Web is proceeding in a step-by-step approachbuilding one layer on top of another. Two important technologies for developingthe Semantic Web are already in place: extensible Markup Language (XML) andthe Resource Description Framework (RDF).

Building one layer upon another requires each layer to have downward compatibityand upward partial understanding. Downward compatibility means thatagents are fully aware of a layer to interpret and use information written at alower level. Upward partial understanding means that agents take at least partialadvantage of information at higher levels. For example, an agent aware ofRDF and RDF Schema semantics can interpret knowledge written in OWL, bydisregarding all but RDF and RDF Schema elements.

Extensible Markup Language lets everyone create their own tags. Scripts, orprograms, can make use of these tags in sophisticated ways, but the script writerhas to know how the page writer uses each tag. In short, XML allows usersto add arbitrary structure to their documents, but says nothing about what thestructure means.

Why is so XML important? Just as HTML is an open standard that allowsinformation exchange and display over the Internet, XML is an open standardthat allows data to be exchanged between applications over the Web. XML is thebridge to exchange data between the two main software development frameworksover the Web: J2EE and .NET. We can consider XML as a highly functionalsubset of SGML, but as a result, it is a meta-language that allows users to designtheir own markup languages.

MODULE 2
Knowledge Representation For The Semantic Web

Ontologies:

Ontology's and their role in the Semantic Web:

The idea of the Semantic Web is to extend unstructuredinformation with machine process able descriptions of the meaning (semantics)of information and to provide missing background knowledge where required.The key challenge of the Semantic Web is to ensure a shared interpretation of information.Related information sources should use the same concepts to reference thesame real world entities or at least there should be a way to determine if two sourcesrefer to the same entities, but possibly using different vocabularies. Ontologies andontology languages are the key enabling technology in this respect. Ontology, byits most cited definition in AI, is a shared, formal conceptualization of a domain,i.e. a description of concepts and their relationships. Ontologies aredomain models with two special characteristics, which lead to the notion of sharedmeaning or semantics:

1. Ontologies are expressed in formal languages with a well-defined semantics.
2. Ontologies build upon a shared understanding within a *community*. This understandingrepresents an agreement among members of the community over theconcepts and relationships that are present in a domain and their usage.

The first point underlines that ontology needs to be modelled using languageswith a formal semantics. RDF and OWL, the ontology languagesthat we introduce later in this Chapter, have standardized syntaxes and logic-basedformal semantics. RDF and OWL are the languages most commonly used on theSemantic Web, and in fact when using the term ontology many practitioners refer todomain models described in one of these two languages. The second point reminds as that there is no such thing as a "personal ontology". For example, the schema ofa database or a UML class diagram that we have created for the design of our ownapplication is not ontology. It is a conceptual model of a domain, but it is notshared: there is no commitment toward this schema from anyone else but us.

Note that the definition does not define the level of detail at which ontologiesneed to be modeled nor does it specify the minimal expressivity required from an ontology language. In practice, ontologies differ greatly in complexity. adapted from the work of Smith and Welty we loosely organize the mostcommon ontological structures according to their complexity.

The simplest structures are glossaries or controlled vocabularies, in essencean agreement on the meaning of a set of terms. (Note that amere bag of words doesn'tmeet the criteria of shared meaning.) An example would be a controlled vocabularyused inside a support center for describing incidents reported. Such a controlled vocabularyfacilitates the communication among the helpdesk and the technical staff ofa support center as it enables a uniform description of the reported problems.

Semantic networks are essentially graphs that show also how terms are related toeach other. Thesauri are richer structures in that they describe a hierarchy betweenconcepts and typically also allow describing related terms and aliases. Thesauri arealso the simplest structures where logic-based reasoning can be applied: the broader narrowerrelationships of these hierarchies are transitive, in that an item that belongsto a narrower category also belongs to its direct parent and all of its ancestors.

In the context of the Semantic Web research, it is often assumed that an ontologyat least contains a hierarchy between the concepts (subclass relationships).

Weaker models with no explicit hierarchies, such as the folksonomiesare often excluded from the definition. However, we will argue in timethat it is possible to extract hierarchies as well as relationships between the tags infolksonomies. Further, folksonomies contain additional information about the socialcontext of tags, i.e. the set of users who have been using them.

The term *lightweight ontology* is typically applied to ontologies that make a distinctionbetween classes, instances and properties, but contain minimal descriptionsof them. On the other hand, *heavyweight ontologies* allow describing more preciselyhow classes are composed of other classes, and provide a richer set of constructs toconstrain how properties can be applied to classes. At the far ends of the spectrum arecomplex knowledge bases that use the full expressivity of first order logic (FOL) todefine to a great detail the kind of instances a concept may have and in which casestwo instances may be related using a certain relationship. The more constrained thedescriptions of concepts are, the less likely that their meaning will be misinterpretedby human readers. Further, the closer to the far end of the spectrum, the larger the

Role that computers can play in reasoning with ontologies. Typical reasoning tasks include checking the consistency of the usage of terms, classifying concepts (findingsubclass relations among concepts) and checking for equivalent terms.

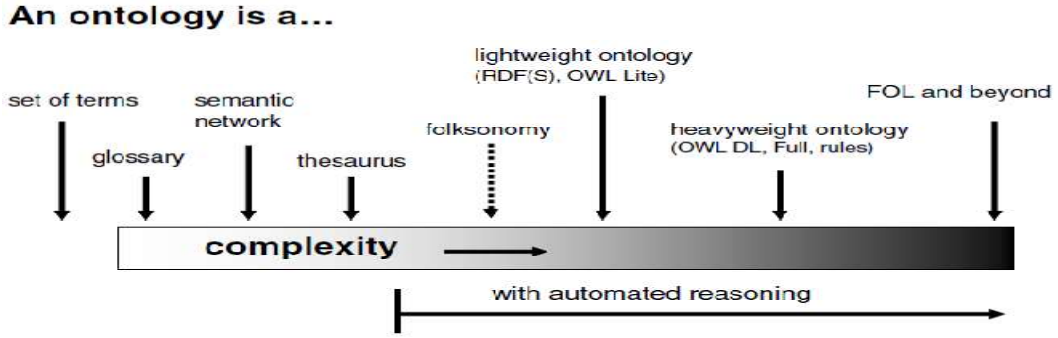

Figure: Ontology's can be organized according to complexity (informally, the level ofsemantics).

Roughly speaking, the models that can be captured using less expressive languagescan also be expressed using more expressive languages, e.g. a modeling languagecapable of describing a semantic network (a graph structure) is typically alsocapable to describe a classification hierarchy (a tree structure). One may thus wonderwhy a single, most expressive language is not enough to satisfy all modelingneeds. The answer is partly that a language that is too expressive (offers features thatare not used) can be in the way of modeling. Just like in the case of programminglanguages, every ontology language offers a slightly different (and not entirely overlapping)set of features that precisely matches the representation needs of a specific

task or domain. We will also see that less expressive modeling languages have lowerguaranteed computational complexity when it comes to reasoning.

The most common Web ontologies are all lightweight ontologies dueto the need of serving the needs of many applications with divergent goals. Widelyshared Web ontologies also tend to be small as they contain only the terms that areagreed on by a broad user base. Large, heavyweight ontologies are more commonlyfound in targeted expert systems used in focused domains with a tradition of formalizedprocesses and vocabularies such as the area of

life sciences and engineering. Wehave argued elsewhere that the trade-off between the formality and sharing scopeof knowledge is unavoidable: models that need to be shared among many applicationswill end up to be shallow, while a limited scope allows to create more detailedmodels of the world

Ontology languages for the Semantic Web

Although the notion of ontologies is independent of the Web, ontologies play a specialrole in the architecture of the Semantic Web. We show a schematic view of thisarchitecture in Figure. This architecture provides the main motivation for the designof ontology languages for the semantic Web: RDF and OWL are both preparedfor the distributed and open environment of the Web.

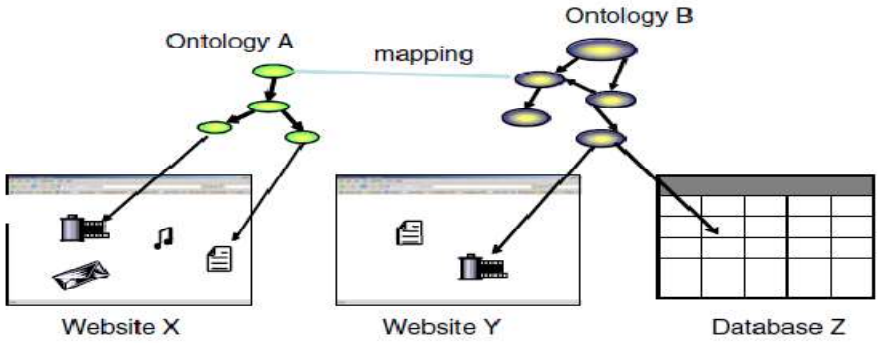

Figure: The Semantic Web in envisioned as a network of ontologies that adds machineprocess ablesemantics to existing web content, including Web resources and databases.

The Semantic Web will be realized by annotatingexisting Web resources with ontology-based metadata and by exposing thecontent of databases by publishing the data and the schema in one of the standardontology languages.

Ontology languages designed for the Semantic Web provide the means to identifyconcepts in ontologies using globally unique identifiers (URIs). These identifiers canbe used in data sources to point to a concept or relationship from an external, publicontology. Similar to creating HTML pages and linking them to existing ones, anyonecan create and publish an ontology, which may reference the concepts in remoteontologies. Much like the hyperlinks among web pages, it is expected that thesereferences will form a contiguous web linking all knowledge sources across the Web.As URLs are also URIs, ontologies can also reference existing Web resources anddescribe their characteristics.

The (meta)data and its schema can be made public either by publishing an RDFor OWL document online (which may be dynamically generated from a database orthrough a Web service invocation) or by providing access to a data source throughimplementing the standard query language and protocol of the Semantic Web. The schema and the data itself can be published as separate documents or query endpoints,but as we will see, ontology languages for the Semantic Web allow mixinginformation about the schema of a data source and the instances (the records, indatabase terms).

Semantic Web applications collect or query such data sources, aggregate andreason with the results. Information described according to a single schema that isknown by the application developer can be directly consumed: by committing to ontology the parties involved have already implicitly agreed on the meaning ofthe information. In this case, the interesting task left is the aggregation of instancedata, i.e. finding equivalent instances across the data sources, if they exist.

When annotating Web content or exposing the content of a database, one maychoose to create ontology from scratch, or reuse an existing ontology while possiblyextending it. However, as there is no coordination of any kind in reusing ontologies,it may happen that two communities develop ontologies that cover the same oroverlapping domains. (For example, there are several known ontologies that describetypes of scientific publications and their properties (e.g. scientific articles that have atitle, a set of authors, a journal and a publication date.) In this case, we need to determinehow the classes in one ontology are related to the classes in ontology.(For example, that the Article class in one ontology is the same as the Journal Publicationclass in ontology.) Finding equivalences and other relations amongterms in different ontologies is the task of *ontology mapping*, a problem we do notaddress in this volume.

What we would like to emphasize is the role of machine-process able semantics inboth the mapping of classes and instances. The more formal and detailed ontologyis, the more accurately it can capture the intended meaning of concepts and relationships.

Ontology languages for the Semantic Web In the following, we introduce the ontology languages RDF and OWL, which havebeen standardized in recent years by the World Wide Web Consortium.

The Resource Description Framework (RDF) and RDF Schema

The Resource Description Framework (RDF) was originally created to describe resourceson the World Wide Web (in particular web pages and other content), hencethe name. In reality, RDF is domain-independent and can be used to model both realworld objects and information resources. RDF itself is a very primitive modelinglanguage, but it is the basis of more complex languages such as OWL.

There are two kinds of primitives in RDF: resources and literals (character sequences).The definition of a resource is intentionally vague; in general everything ismodeled as a resource that can be (potentially) identified and described. Resourcesare either identified by a URI or left blank. URIs are identifiers with a special syntaxdefined in Blank resources (*blank nodes*) are the existential quantifiersof the language: they are resources with an identity, but whose identifier is notknown or irrelevant. (This is no way in the RDF model itself to assign an identifierto a blank node.) Literals are strings (character literals) with optional language anddata type identifiers.

Expressions are formed by making statements (*triples*) of the form (subject, predicate,and object). The subject of a statement must be a resource (blank or with a URI),the predicate must be a URI and the object can be either kind of resource or a literal.Literals are thus only allowed at the end of a statement.

RDF is very easy to understand in practice. The following brief RDF documentdescribes a person named Rembrandt. The six statements are alsoshown as a directed, labeled graph. In this visualization the nodes arethe subjects and objects of statements, labeled with the URI or literal or left blank,and the edges connect the subjects and objects of statements and are labeled withthe URI of the property. As we can see the

statements of the RDF model form agraph because the object of one statement can be the subject of another statement.(As noted literals cannot be the subjects of statements, i.e. there are no arrows goingfrom literals.)

This fragment is written in the Turtle syntax, one of the many notations of RDFgraphs . We will later also encounter the RDF/XML syntax, an XML-based notationdefined by theWorldWideWeb Consortium. RDF/XML has been the first and up to date the most widely used syntax. As an XML based syntax, it alsohas the advantage of being process able by XML tools. However, other notations suchas Turtle are often preferred for their readability by human authors.

A set of triples describing two persons represented in the Turtle language.

@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#label> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix example: <http://www.example.org/> .
example:Rembrandtrdf:typefoaf:Person .
example:Saskiardf:typefoaf:Person .
example:Rembrandtfoaf:name "Rembrandt" .
example:Rembrandtfoaf:mbox<mailto:rembrandt@example.org> .
example:Rembrandtfoaf:knowsexample:Saskia .
example:Saskiafoaf:name "Saskia" .

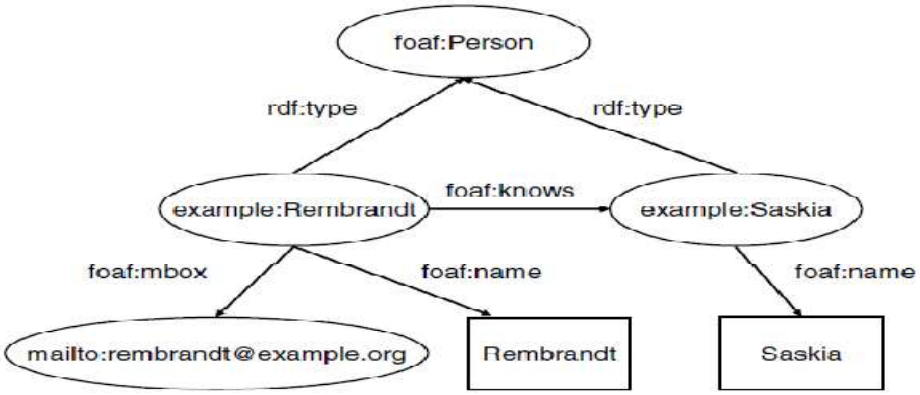

Figure:A graph visualization of the RDF document

The RDF language provides the basic term to assign a type to a resource(*rdf*:*type*) and to declare a resource as a property . It also provides features to describe collections of instances12 and to make statements about statements (*reification*)13. The descriptive power of RDF is minimal that in practice it is always usedin combination with RDF Schema. RDF Schema is a simple extension of RDF defininga modeling vocabulary with notions of classes and subclasses.

Classes and properties can be connected by specifying the domain and range of properties.Note that the division of terms between the RDF and RDF Schema namespaces isvery unnatural: for example, *rdf*:*type*property appears in the RDF namespace, eventhough there are no classes in RDF. On the other hand, the *rdfs*:*Literal*class is in theRDF(S)

namespace even though literals are a feature of RDF. For this reason and forthe limited use of RDF on its own, most people refer to an RDF Schema ontologywhen talking about an 'RDF ontology' or 'RDF data'. In the following, we will alsouse the term RDF in this sense and apply the term RDF Schema when we specificallyintend to speak about the RDF Schema vocabulary.

The firstgroup of statements describes the class Person. The type of the resource is specifiedas *owl*: *Class*, we give a label, name a super class and state that this class is disjointfrom the class of documents. We then describe the *foaf: knows* and *foaf: name* properties we haveBasic constructsused above, specifying their type label, domain, range and a super property in thecase of *foal* :*name*.

What is likely to be surprising here is that in comparison to most other modelingFrameworks we use the same model of subject/predicate/object to describe instancesas well as classes. The statements describing a set of instances and theStatements describing the higher level of classes are typically stored inSeparate documents on the Web, but this is not a requirement.

In fact it is not always trivial to separate the description of the instances from thedescription of classes. We may want to store instance data and the ontology togetherfor practical reasons. But also, for example, we may want to describe a class usingtypical instances such as when describing the concept of a Weekday, which is a classof five days. In this case drawing a clear line between instance data (metadata) andclass data (schema) is problematic even from a conceptual point of view. This isreflected in the rather ambiguous usage of the term ontology, which is sometimesused to refer to the classes only, and sometimes to a set of instances and classes bundled together.

The example also reveals some of the impressive flexibility of RDF. The constructsof language itself form a vocabulary just like the terms of any domain ontology.In other words, terms like *rdfs*:*Class*are not treated any different from user definedclasses. This means that it is very well possible to form statements aboutthe elements of the language itself. This is used sometimes to establish relationshipbetween elements of a domain and the RDF Schema vocabulary, for example to statethat a domain specific labeling property (such as a property providing the name fora person) is a sub property of the more general *rdfs*: *label* property. We have done thatwhen declaring the *foaf: name* property as a sub property of the *rdfs*:*label*property.

Some statements from the FOAF ontology about the terms used in the previousexample.

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
foaf:Personrdf:typeowl:Class .
foaf:Personrdfs:label "Person" .
foaf:Personrdfs:subClassOffoaf:Agent
foaf:Personowl:disjointWithfoaf:Document .
foaf:knowsrdf:typeowl:ObjectProperty .
foaf:knowsrdfs:label "knows" .
foaf:knowsrdfs:domainfoaf:Person .
foaf:knowsrdfs:rangefoaf:Person .
foaf:namerdf:typeowl:DatatypeProperty .
foaf:namerdfs:label "name" .
foaf:namerdfs:subPropertyOfrdfs:label .
foaf:namerdfs:domainowl:Thing .
```

foaf:namerdfs:rangerdfs:Literal

Although it makes little sense, one might even state for example that *rdfs*:*Resource*is an *rdfs*:*subClassOfrdfs*:*Class*, effectively claiming that all resources are classesin the model.Further, RDF makes no clear separation between classes, instances and properties.One can create classes of instances (met modeling), which is often required inmodeling practical knowledge. (The classical example is modeling the notion ofspecies as a class of classes of animals.) Metamodeling is also present in the example,as we have seen that the *rdf*:*type*is used on both an instance (Rembrandt,whose class is Person), and a class (Person, whose class is the class of all classes)and we used the *rdfs*:*label*property on both instances, classes and properties. Classesof properties are also often used to characterize properties. For example, in the caseof the OWL language the resource *owl*:*DataTypeProperty*is the class of propertiesthat have Literal values.

Lastly, part of the flexibility of RDF is that language constructs are not interpretedas strictly as we might expect. For example, even though the range of the*foaf*:*knows*property is defined to be *foaf*:*Person* we can still add the statementthat *example*:*Rembrandtfoaf*:*knows mailto*:*saskia@example.org*. While one wouldmight expect that this leads to a logical contradiction as we are confusing peopleand their email addresses, this RDF statement is merely interpreted as a hint that theresource *mailto*:*saskia@example.org* is both an email address *and* a Person.

RDF and the notion of semantics

In the above we have been largely concerned with the syntax of the language: thekind of symbols (resources and literals) we have and the way to form statementsfrom them. Further, we introduced some special resources that are part of the RDFand RDF Schema languages (e.g. *rdfs*:*subClassOf*) and gave their meaning in a colloquialstyle. However, if we want to use RDF(S) for conveying knowledge across theWeb, we need to be able to define the meaning of our language (and thus the meaningof ontologies) in a much more reliable fashion. This is important from the perspectiveof machine reasoning: for example, we are often interested to check whether a certainstatement necessarily follows from a set of existing statements or whether thereis contradiction among a set of statements. What we then need is an agreement on amethod to unambiguously answer such questions independently of our interpretationof the natural language description of the RDF(S) constructs.

The meaning of RDF(S) constructs is anchored in a model-theoretic semantics,one of the most common ways to provide semantics [Hay04]. Using model-theoreticsemantics meaning is defined by establishing a mapping from one model to a metamodelwhere the truth of propositions is already uniquely determined. In the contextof RDF such a mapping is called an *interpretation*.

Although a meta-model can be defined in another formal system, it is convenientto think of the meta model as some independently existing reality that the ontology isintended to describe. Thus an interpretation can be thought of as a mapping betweensymbols and the objects or relations they intended to describe.

Figure: An interpretation is a mapping between the terms of ontology and an interpretationdomain.

The constructs of RDF are used to put constraints on possible interpretations andto exclude thereby some of the unintended interpretations. For example, we onlywant to allow interpretations where symbols for the two persons are mapped to differentobjects. In other words, we want to exclude interpretations where the twoinstances are mapped to the same object. In order to achieve this we can specifythat the instance Rembrandt is *owl*:*differentFrom*the second instance Saskia.

The model theory of RDF(S) provides some axiomatic triples that are true inevery RDF(S) interpretation (for example, *rdf*:*Propertyrdf*:*type rdfs*:*Class*) and definesthe constraints that language elements put on possible interpretations of RDF(S)models. In practice, these semantic conditions can be expressed as a set of rules: forexample, the semantics of *rdfs*:*subPropertyOf*is given by the following rules:
(*aaa, rdfs*:*subPropertyOf , bbb*) (*bbb, rdfs*:*subPropertyOf , ccc*)
→ (*aaa, rdfs*:*subPropertyOf , ccc*)
(*aaa, rdfs*:*subPropertyOf , bbb*) → (*aaa, rdf*:*type, rdf*:*Property*)
(*aaa, rdfs*:*subPropertyOf , bbb*) → (*bbb, rdf*:*type, rdf*:*Property*)
(*xxx, aaa, yyy*) (*aaa, rdfs*:*subPropertyOf , bbb*) → (*xxx, bbb, yyy*)

The most noteworthy feature of RDF semantics is that the interpretation of thelanguage is based on an open world assumption and is kept monotonic. An openworld assumption means that based on a single document we cannot assume that wehave a complete description of the world or even the resources explicitly describedtherein. Monotonicity means additional knowledge added to an RDF knowledge basecannot make previous inferences invalid.17 For example, if we specify that the rangeof the foaf:knows property is Person and then state that Rembrandt knows an instanceof another class such as Pluto the dog (or even a literal value) we do not cause a(logical) contradiction; it is assumed that there could exist a statement defining thatsome other class (e.g. Dog) is also in the range of *foaf*:*knows*.

A consequence of monotonicity is that it makes no sense of talking about RDFvalidation: RDF Schema statements about an RDF resource only add additional informationabout the resource and cannot invalidate or contradict statements previouslyinferred about that resource. In fact, RDF Schema semantics is specified as aset of inference rules; in the example, these rules would allow to infer that the resourceprovided as the object of the *foaf*:*knows*property is (also) a Person besidespossibly being other things

SPARQL: querying RDF sources across the Web

Through the years several ontology databases (triple stores) have been developed,each featuring their own query language. As all of these query languages operate onthe same data model (an RDF graph), these query languages have in common thatthey allow to select retrieve parts of an RDF graph by matching a graph pattern providedby the query and return the possible values of variables in the query pattern. There are, however, differences in syntax and the exact set of features.

At the time of writing, theWorldWideWeb Consortium is working on creating a recommendation called SPARQL, establishing a standard query language and a protocolfor interacting with RDF sources. The query language captures the commonset of

features of the existing RDF query languages [PS06]. The protocol prescribesthe way a software application would query a remote ontology store across the Internet,i.e. the way to submit a SPARQL query to a server and the expected formatof the results (and eventual error messages) [Cla06]. The SPARQL specifications areexpected to significantly increase the interoperability of SemanticWeb applications.Also, the content of non-RDF databases may also be exposed using a SPARQL interface,opening up their content to the SemanticWeb world. (The process of wrappinga relational database in a SPARQL interface can be partially automated.

Web languages:The Web Ontology Language (OWL)

The Web Ontology Language (OWL) was designed to add the constructs of DescriptionLogics (DL) to RDF, significantly extending the expressiveness of RDFSchema both in characterizing classes and properties. Description Logics are a set ofKnowledge Representation languages with formal semantics based on their mappingto First Order Logic (FOL). Description Logics have been extensively studied sincethe 1980s including studies on the tradeoffs between the expressivity of the chosen language and the efficiency of reasoning. OWL has been designed in a way that itmaps to a well-known Description Logic with tractable reasoning algorithms.

The Web Ontology Language is in fact a set of three languages with increasingexpressiveness: OWL Lite, OWL DL and OWL Full. These languages are extensionsof each other ($OWLLite \subseteq OWLDL \subseteq OWLFull$) both syntactically and semantically.For example, every OWL Lite document is a valid OWL DL document andhas the same semantics when considered as an OWL DL document, e.g. it leads tothe same logical conclusions. The vocabularies of these languages extend each otherand languages further up in the hierarchy only relax the constraints on the use ofthe vocabulary. Although it is generally believed that languages of the OWL family
Would be an extension of RDF(S) in the same sense, this is only true for OWL Full,the most expressive of the family ($RDF(S) \subseteq OWLFull$).

The middle language, OWL DL was the original target of standardization and itis a direct mapping to an expressive Description Logic. This has the advantage thatOWL DL documents can be directly consumed by most DL reasonless to performinference and consistency checking. The constructs of OWL DL are also familiar,although some of the semantics can be surprising mostly due to the open world assumption. (Shows the OWL DL vocabulary, which is the sameas the vocabulary of OWL Full.) Description Logics do not allow much of the representationflexibility introduced above (e.g. treating classes as instances or definingclasses of properties) and therefore not all RDF documents are valid OWL DL documentsand even the usage of OWL terms is limited.

For example, in OWL DL it is not allowed to extend constructs of the language,i.e. the concepts in the RDF, RDF Schema and OWL namespaces. In the case of thenotion of a Class, OWL also introduces a separate *owl*:*Class*concept as a subclass of*rdfs*:*Class*in order to clearly distinguish its more limited notion of a class. Similarly,OWL introduces the disjoint classes of object properties and datatype properties. Thefirst refers to properties that take resources as values (such as *foaf*:*knows*) and thelatter is for properties ranging on literals such as *foaf*:*name*.

OWL Full is a "limitless" OWL DL: every RDF ontology is also a valid OWLFull ontology and has the same semantics when considered as an OWL Full document.However,

OWL Full is undecidable, which means that in the worst case OWLFull reasoners will run infinitely. OWL Lite is a lightweight sub-language of OWLDL, which maps to a less expressive but even more efficient DL language. OWL Litehas the same limitations on the use of RDF as OWL DL and does not contain someof the terms of OWL DL.

In summary, RDF documents are not necessarily valid OWL Lite or OWL DLontologies despite the common conviction. In fact, "downgrading"a typical RDF or OWL Full ontology to OWL DL is a tedious engineeringtask. It typically includes many simple steps such as declaring whether properties areobject properties or data type properties and importing the external ontologies usedin the document, which is mandatory in OWL but not in RDF. However, the process often involves more fundamental modeling decisions when it comes to finding alternaterepresentations.

Most existing web ontologies make little use of OWL due to their limited needs,but also because general rule-based knowledge cannot be expressed in OWL. Theadditional expressivity of OWL, however, is required for modeling complex domainssuch as medicine or engineering, especially in supporting classification taskswhere we need to determine the place of a class in the class hierarchy based on itsdescription.

Unified Modeling Language (UML)

UML is most commonly used in the requirements specification and design of object-orientedsoftware in the middle tier of enterprise applications [Fow03]. The chief difference between UML and RDF(S)/OWL is their modeling scope: UML containsmodelingprimitives specific for a special kind of information resource, namely objectsin an information system characterized by their static attributes and associations,but also their dynamic behavior. Many of the modelling primitives of UMLare thus specific to objects and their role in OO systems; interfaces, functions etc.are examples of such constructs.22 Nevertheless, if we ignore these constructs ofthe languages and the difference in scope, there is still a significant overlap in theexpressiveness of object-oriented models of a domain and ontological models.

Figure 4.8 shows our examplemodelled in UML. It is natural to model propertiesthat take primitive values as datatypes and model all other properties as associations.(However, attributes can also take model classes as types.) UML is primary a schemadefinition language and thus the modelling of instances is limited.

Based on the comparison of OWL Full and UML 2.0 we can note that there is asignificant overlap as well as differences in the modelling capabilities of OWL andUML [HEC+04]. In the following we summarize the more specific differences bylooking at the unique features of these frameworks.

Comparison to the Extensible Markup Language (XML) and XMLSchema

Up to date XML is the most commonly used technology for the exchange of structuredinformation between systems and services. From all languages discussed therole of XML is thus the most similar to RDF in its purpose.

The most commonly observed similarity between XML and RDF is a similaritybetween the data models: a directed graph for RDF and a directed, ordered tree forXML. In XML, the tree is defined by the nesting of elements starting with a singleroot node. This model originates from the predecessor of XML called SGML whichwas primarily used for marking up large text documents. Text documents follow thetree structure themselves as paragraphs are nested in subsections, subsections arenested in sections, sections are nested chapters etc. The ordering of the children ofan element matters, which is again inherited from the text

processing tradition. Onthe other hand, RDF proposes a more relaxed data model based on arbitrary directedgraphs built from single edges between the nodes representing classes or instances.This structure is better suited for creating conceptual domain models, which are oftenso rich in cross-taxonomical relationships that a hierarchical representation wouldprove to be difficult. Noting that every tree is a graph (and ignoring the ordering of children for a moment) one would expect that it is trivial to represent XML modelsin RDF, although as we will see this is not the case.

Much like RDF, XMLitself is merely a common conceptual model and syntax fordomain specific languages each with their own vocabulary (hence the name extensible).Examples of these languages include XHTML, but also specialized languagessuch as the GraphML language for descriptions of graphs or the SVG image formatfor vector graphic images. XML has a number of schema languages such as XMLSchema and Relax NG to define such languages. The use of these schema languages,however, is radically different. Namely, schemas written in XML schema languagesnot only define the types of elements and their attributes but also prescribe syntax i.e. the way elements are allowed to be nested in the tree. XML documents can be validatedagainst a schema on a purely syntactic level. Schema languages for RDF (RDFSchema and OWL) do not impose constraints directly on the graph model but effectthe possible interpretations of metadata. Consistency of an RDF/OWL model ischecked by searching for possible interpretations. (If there are no possible interpretationsof a model than it is inconsistent.) As we have already seen, RDF and OWL alsotranscend the relatively vague notions of XML (such as elements, attributes, nestingetc.) and provide a more refined set of constructs (classes, instances and properties).

To illustrate the difference in the focus of XML and RDF, let us consider how wewould represent the same information captured in Figure 4.3 in XML. Figure 4.10shows three possible XML representations. While the intended meaning (semantics)of these documents is the same, the syntax and thus the resulting XML tree is different.(The order of the branches matters in XML, which is the only differencebetween the second and third examples.) This means that applications working onthe basis of different representations would be incompatible: XML queries and XSLtransformations performed on these different representations would produce differentresults. Thus there needs to be an agreement on both syntax and semantics whenusing XML. For web-based data interchange RDF has a clear advantage here in thatagreement on a shared XML format would require a much stronger commitmentthan the agreement of using RDF. The kind of agreement that is needed to exchangeRDF documents concerns only the representation of individual statements (the simplesubject/predicate/object model) as we have seen on the example of the Turtlelanguage.

As an aside, we also see the difficulty of XML in dealing with graph-like structures:choosing the top node is an arbitrary decision when dealing with such information.This is particularly obvious in representing social networks (Person entitieslinked with knows relationships), it is generally true that real world domain modelscan be rarely forged into a single, unique tree structure.

XML is highly appreciated for the variety of complementary tools and technologiessuch as XML databases, standard schema, query and transformation languages(XQuery and XSLT), a range of editors, parsers, processors etc. The legacy of XMLcompelled also theW3C to adopt it also as a notation for RDF and OWL. This kind ofcontingency is popularized by the famous Semantic Web layer cake diagram in Figure4.1128 This picture

shows the vision of the Semantic Web as a set of languagesbuilding upon existing standards such as XML, URIs and Unicode. Despite the goodintentions, however, this diagram obfuscates the true relationship between XML andontology languages from the W3C such as RDF and OWL. (We have already seenthat the relationship between RDF and OWL is also more complicated than simple.
Three different XML representations of the same information.extension.) By doing so it has done more damage to the argument for ontology languagesover XML than any other conceptualization of the next generation Web.

```
<Person name="Rembrandt">
<mbox>mailto:rembrandt@example.org</mbox>
<knows>
<Person name="Saskia" />
</knows>
</Person>
<Person name="Rembrandt">
<mbox>mailto:rembrandt@example.org</mbox>
<knows>
<Person ID="1"/>
</knows>
</Person>
<Person name="Saskia" ID="1" />
<Person name="Saskia" ID="1" />
<Person name="Rembrandt">
<mbox>mailto:rembrandt@example.org</mbox>
<knows>
<Person ID="1"/>
</knows>
</Person>
```

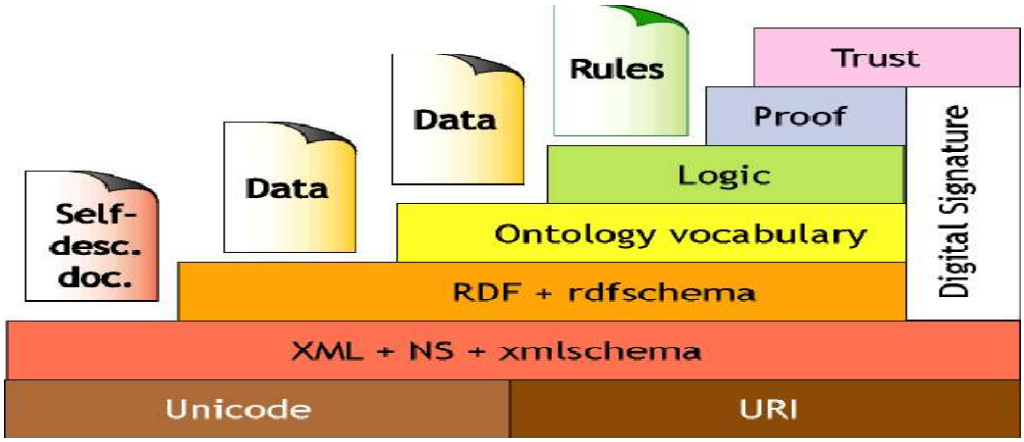

Figure: The Semantic Web layer cake.

The layer cake suggests, namely that RDF builds on XML is true onlyto the extent that RDF models can be exchanged using an XML format named RDF/XML.29 In the RDF/XML syntax the statements from the graph are simplywritten out one-by-one with some possibilities for abbreviation. This means thatthere are a number of possible XML serializations for every RDF graph dependingon whether these abbreviations are used, how the statements are ordered etc.In practice, while such RDF/XML documents are valid XML and thus they can bestored in XML databases and queried using XML Query languages such as XQueryor transformed using XSLT, these queries and transformations will be sensitive to aparticular serialization. All in all, the only useful tools with respect to RDF/XMLfiles are XML editors and they are only helpful in checking the well-forkedness ofthe representation.

In the other direction, the re-representation of XML schemas and data using RDFis also more difficult than suggested by the layer cake. Transforming XML documentsminimally requires assigning namespaces to elements and attributes. Further,RDF/XML mandates a striped syntax requiring that instances and propertiesare nested correctly into each other. Most XML schemas, however, have not beendesigned to conform with this striped syntax. For example, the following XML documentfrom the XML Schema Primer violates the assumption of the striped syntaxin that it lacks a level of elements between shipTo and name: name is a property aninstance of an omitted Person class.

```
<?xml version="1.0"?>
<apo:purchaseOrderxmlns:apo="http://www.example.com/PO1"
orderDate="1999-10-20">
<shipTo country="US">
<name>Alice Smith</name>
<street>123 Maple Street</street>
<!-- etc. -->
</shipTo>
<billTo country="US">
<name>Robert Smith</name>
<street>8 Oak Avenue</street>
<!-- etc. -->
</billTo>


<apo:comment>Hurry, my lawn is going wild</apo:comment>
<!-- etc. -->
</apo:purchaseOrder>
```

While there is no way to automate the process of XML to RDF conversion inthe general case, it is possible to design XSLT transformations that carry out this jobfor particular schemas. The rather inaptly named GRDDL30 specification provides astandard way of attaching such transformations to XML documents. GRDDL onlydescribes an agreed way to attach transformations to an XML or XHTML document:the agreement on the way to represent and extract metadata is left to the authorsof XML formats and transformations. (For example, there is a style sheet to extractFOAF data from appropriately marked up XHTML documents.31 AnotherW3C proposal,the RDF/A syntax gives a generic, domain-

independent way to embed RDFmetadata in XHTML by adding certain attributes and special elements [AB06].

For newly designed formats a practical alternative is to create a DTD or XMLSchema that takes into account the requirements of RDF/XML or put differently:use RDF/XML in a constrained way so that it conforms to a particularXML Schema.This is the approach that was used in the design of the RSS 1.0 format, which can bemeaningfully processed by both XML and RDF tools.

Transforming existing XML Schema documents into RDF/OWL is unfortunatelynot a trivial task, because the XML model and the rather complex XML Schemastandard have notions that RDF/OWL lack and vice versa. For example, XML distinguishesbetween attributes and terminal elements with simple type values (the differencebetween the attribute country and the element name in the above example)and there is a natural ordering of elements32, i.e. the schema for the above examplecan specify that name has to be listed before email while in RDF the order in whichproperties are listed cannot be constrained. The advanced features of XML Schemainclude the definition of uniqueness which is similar to the notion of keys in E/Rmodels. There is also a possibility to define keys and key references linking parts ofthe schema through an integrity constraint, i.e. that the key reference should containan existing value from the set of keys. These constructs can not be trivially expressedin OWL either.

MODULE III
Ontology Engineering

Ontology Devolpment: The field of Philosophy originally defined the word ontology to represent the concept of existence. It is the theory of objects and their interrelationships. As used in information science, the term ontology frequently refers to a hierarchical data structure containing the relevant objects and their relationships, as well as the rules within that domain.

In the field of Artificial Intelligence (AI), ontology applications have been developed for knowledge management, natural language processing, e-Commerce, education, and new emerging technologies such as the Semantic Web. The Semantic Web requires the construction of ontologies for its various representation languages, query languages, and inference technologies. The basic methodology for designing and building ontologies. In addition, ontology matching and mapping, which are essential to knowledge representations, are described.

ONTOLOGY ENGINEERING

Ontology is the formal specification of terms within a domain and their relationships. It defines a common vocabulary for the sharing of information that can be used by both humans and computers. Ontologies can be in the form of lists of words; taxonomies, database schema, frame languages and logics. The main difference between these forms is their expressive power. Ontology together with a set of concept instances constitutes a knowledge base.

If a program is designed to compare conceptual information across two knowledge bases on the Web, it must know when any two terms are being used to mean the same thing. Ideally, the program must have a way to discover common meanings for whatever knowledge bases it encounters. Typically, an ontology on the Web will combine a taxonomy with a set of inference rules.

Taxonomy is defined as a set of classes of objects and their relationships. These classes, subclasses, and their relationships are important tools for manipulating information. Their relations are described by assigning properties to classes and allowing subclasses to inherit these properties. Ontology then is a taxonomy plus inference.

Ontology inference rules allow manipulation of conceptual information. The most important ontology relationship is the subsumption link (e.g., subtype and super type link).

When a network of concepts is represented by a tree, it rigorously defines the taxonomy. While ontology can sometimes be modularized as a set of trees, some advocate that all ontology should be taxonomic, but others favor a lattice structure. For example, ontology rigorously defines a Thesaurus structure when it uses the related-to link in addition to the subsumption link

Ontology engineering seeks a common vocabulary through a data collection process that includes discussions, interviews, document analysis, and questionnaires. Existing ontologies on a subject are discovered, assessed, and reused as much as possible to avoid "reinventing the wheel." As part of this process, ontologies are designed as living objects with a maintenance cycle.

Ontology Applications

The simplest ontology consists of a simple taxonomy with a single relation. Categories of ontology applications can be grouped as • Neutral Authoring: The author of an object in a single language translates into a different format for use in alternative applications.

• Ontology as Specification: Ontology of a given domain is created and used as a basis for specification and development of some software. This approach allows documentation, maintenance, reliability and knowledge (re)use.

• Common Access to Information: Information in an inaccessible format becomes intelligible by providing a shared understanding of the terms, or by mapping between sets of terms.

Ontology-Based Search: Ontology is used for searching an information repository.

CONSTRUCTING ONTOLOGY

Ontology permits sharing common understanding of the structure of information among people and software agents. Since there is no unique model for a particular domain, ontology development is best achieved through an iterative process.

Objects and their relationships reflect the basic concepts within an ontology. An iterative approach for building ontologies starts with a rough first pass through the main processes as follows:

• First, set the scope. The development of an ontology should start by defining its domain and scope. Several basic questions are helpful at this point: What will the ontology cover? How will the ontology be used? What questions does the ontology answer? Who will use and maintain the ontology? The answers may change as we proceed, but they help limit the scope of the model.

• Second, evaluate reuse. Check to see if existing ontologies can be refined and extended. Reusing existing ontologies will help to interact with other applications and vocabularies. Many knowledge-representation systems can import and export ontologies directly for reuse.

• Third, enumerate terms. It is useful to list all terms, what they address, and what properties they have. Initially, a comprehensive list of terms is useful without regard for overlapping concepts. Nouns can form the basis for class names, and verbs can form the basis for property names.

• Fourth, define the taxonomy. There are several possible approaches in developing a class hierarchy: a top-down process starts by defining general concepts in the domain. A bottom-up development process starts with the definition of the most specific classes, the levels of the hierarchy, with subsequent grouping of these classes into more general concepts. A combination development process combines the top-down and bottom-up approaches: define the more salient concepts first and then generalize them appropriately.

• Fifth, define properties. The classes alone will not provide enough information to answer questions. We must also describe the internal structure of concepts. While attaching properties to classes one should establish the domain and range. Property constraints (facets) describe or limit the set of possible values for a frame slot.

• Sixth, define facets. Up to this point the ontology resembles a RDFS without any primitives from OWL. In this step, the properties add cardinality, values, and characteristics that will enrich their definitions.

• Seventh, the slots can have different facets describing the value type, allowed values, the number of the values (cardinality), and other features of the values. Slot cardinality: the number of values a slot has. Slot value type: the type of values a slot has. Minimum and maximum value: a range of values for a numeric slot. Default value: the value a slot has unless explicitly specified otherwise.

Eighth, define instances. The next step is to create individual instances of classes in the hierarchy. Defining an individual instance of a class requires choosing a class, creating an individual instance of that class, and filling in the slot values.

• Finally, check for anomalies. The Web-Ontology Language allows the possibility of detecting inconsistencies within the ontology. Anomalies, such as incompatible domain and range definitions for transitive, symmetric, or inverse properties may occur.

Ontologies can be constructed by iterating through this process.

ONTOLOGY DEVELOPMENT TOOLS

Below is a list of some of the most common editors used for building ontologies:

• DAG-Edit provides an interface to browse, query and edit vocabularies with a DAG data structure: http://www.geneontology.org/#dagedit.

• Protege 2000 is the most widely used tool for creating ontologies and knowledge bases: http://protege.stanford.edu/index.shtml.

• WonderTools is an index for selecting an ontology-building tool: http:// www.swi.psy.uva.nl/wondertools/.

• WebOnto is a Java applet coupled with a Web server that allows users to browse and edit knowledge models.

ONTOLOGY METHODS

Several approaches for developing ontologies have been attempted in the last two decades. In 1990, Lenat and Guha proposed the general process steps. In 1995, the first guidelines were proposed on the basis of the Enterprise Ontology and the TOVE (TOronto Virtual Enterprise) project. At the 12th European Conference for Artificial Intelligence in 1996, a method to build ontology in the domain of electrical networks was proposed. The methodology Meth ontology appeared at about the same time. A few years later, the On-To-Knowledge methodology was developed.

The Cyc Knowledge Base (see http://www.cyc.com/) was designed to accommodate all of human knowledge and contains about 100,000 concept types used in the rules and facts encoded in its knowledge base. The method used to build the Cyc consisted of three phases. The first phase manually codified articles and pieces of knowledge containing common sense knowledge implicit in different sources. The second and third phase consisted of acquiring new common sense knowledge using natural language or machine learning tools.

The Electronic Dictionary Research (ERD) project in Japan has developed a dictionary with over 400,000 concepts, with their mappings to both English and Japanese words. Although the EDR project has many more concepts than Cyc, it does not provide as much detail for each one WordNet is a hierarchy of 166,000 word form and sense pairs. WordNet does not have as much detail as Cyc or as broad coverage as EDR, but it is the most widely used ontology for natural language processing, largely because it has long been easily accessible over the Internet Cyc has the most detailed axioms and definitions; it is an example of an axiomatized or formal ontology. Both EDR and WordNet are usually considered terminological ontologies. The difference between a terminological ontology and a formal ontology is one of degree: as more axioms are added to a terminological ontology, it may evolve into a formal or axiomatized ontology.

The main concepts in the ontology development include: a top-down approach, in which the most abstract concepts are identified first, and then, specialized into more specific concepts; a bottom-up approach, in which the most specific concepts are identified first and

then generalized into more abstract concepts; and a middle-out approach, in which the most important concepts are identified first and then generalized and specialized into other concepts.

Methontology was created in the Artificial Intelligence Lab from the Technical University of Madrid (UPM). It was designed to build ontologies either from scratch, reusing other ontologies as they are, or by a process of reengineering them. The Methontology framework enables the construction of ontologies at the knowledge level. It includes the identification of the ontology development process, a life cycle based on evolving prototypes, and particular techniques to carry out each activity. The ontology development process identifies which tasks should be performed when building ontologies (scheduling, control, quality assurance, specification, knowledge acquisition, conceptualization, integration, formalization, implementation, evaluation, maintenance, documentation, and configuration management).

The life cycle identifies the stages through which the ontology passes during its lifetime, as well as the interdependencies with the life cycle of other ontologies. Finally, the methodology specifies the techniques used in each activity, the products that each activity outputs, and how they have to be evaluated. The main phase in the ontology development process using the Methontology approach is the conceptualization phase.

By comparison, the On-To-Knowledge methodology includes the identification of goals that should be achieved by knowledge management tools and is based on an analysis of usage scenarios. The steps proposed by the methodology are kickoff: where ontology requirements are captured and specified, competency questions are identified, potentially reusable ontologies are studied, and a first draft version of the ontology is built; refinement: where a mature and application oriented ontology is produced; evaluation: where the requirements and competency questions are checked, and the ontology is tested in the application environment; and finally ontology maintenance.

ONTOLOGY SHARING AND MERGING

Knowledge representation is the application of logic and ontology to the task of constructing automated models. Each of the following three fields contributes to knowledge representation:

• Logic: Different implementations support different subsets and variations of logic. Sharing information between implementations can usually be done automatically if the information can be expressed a common subset.

• Ontology: Different systems may use different names for the same kinds of objects; or they may use the same names for different kinds.

• Computation: Even when the names and definitions are identical, computational or implementation side effects may produce different behaviors in different systems. In some implementations, the order of entering rules may have inferences that impact computations. Sometimes, the side effects may cause an endless loop.

Although these three aspects of knowledge representation pose different kinds of problems, they are interdependent. Standardizing the terminology used to classify and find the information is important. For artificial intelligence, where the emphasis is on computer processing, effort has been directed to precise axioms suitable for extended computation and deduction.

ONTOLOGY LIBRARIES

Scientists should be able to access a global, distributed knowledge base of scientific data that appears to be integrated, locally available, and is easy to search. Data is obtained by multiple instruments, using various protocols in differing vocabularies using assumptions that may be inconsistent, incomplete, evolving, and distributed. Currently, there are existing ontology libraries including

• DAML ontology library (www.daml.org/ontologies).
• Ontolingua ontology library (www.ksl.stanford.edu/software/ontolingua/).
• Prot´eg´e ontology library (protege.stanford.edu/plugins.html). Available upper ontologies include
• IEEE Standard Upper Ontology (suo.ieee.org).
• Cyc (www.cyc.com).Available general ontologies include
• (www.dmoz.org).
• WordNet (www.cogsci.princeton.edu/~wn/).
• Domain-specific ontologies.
• UMLS Semantic Net.
• GO (Gene Ontology) (www.geneontology.org).
• Chemical Markup Language, CML.

ONTOLOGY MAPPING

Ontology mapping enables interoperability among different sources in the Semantic Web. It is required for combing distributed and heterogeneous ontologies. Ontology mapping transforms the source ontology into the target ontology based on semantic relations. There are three mapping approaches for combing distributed and heterogeneous ontologies:

1. Mapping between local ontologies.
2. Mapping between integrated global ontology and local ontologies.
3. Mapping for ontology merging, integration, or alignment.

Ontology merge, integration, and alignment can be considered as ontology reuse processes.

Ontology merge is the process of generating a single, coherent ontology from two or more existing and different ontologies on the same subject. Ontology integration is the process of generating a single ontology from two or more differing ontologies in different subjects. Ontology alignment creates links between two original ontologies.

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. 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.

There are two forms of deductively valid argument: modus ponens (Latin for "the affirming mode") and modus tollens (the denying mode). 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:

$$\text{RepeatCustomer}(X) \rightarrow \text{discount}(25\%)$$

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.

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:

$$[(p \rightarrow q) \wedge p] \rightarrow [q]$$

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

$$[(p \rightarrow q) \wedge \neg q] \rightarrow [\neg p]$$

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*."

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

$[(p) \wedge (q)] \rightarrow [p \wedge q]$

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

$$[\neg \neg p] \leftrightarrow [p]$$

Tautology is represented as

$[p] \leftrightarrow [p \vee p]$

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 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 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 reasoned otherwise they will cease to trust the results. Doubting unexpected results makes a reasoned useless. 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

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.

Full First-Order Logic Inference Engines

Using full first-order logic for specifying axioms requires a full-fledged automated theorem proved. First-order logic is semi decidable and inference 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 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.

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 maker of computer are defined in the 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 sub graph. 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 sub graph is treated differently from nonfuel subgraphs.

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 sub graph matching and embedded rules by the term "sub graph matching with rules." The unification algorithm divides the sequence of RDF statements into sets where each set constitutes a connected sub graph. This is called a triple set 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 triple set 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 inference 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.

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 need and the information sources available.

MODULE IV

Semantic web applications,services and technology

Semantic web services:

Web Services are self-contained, self-described, component applications that can be published, located, and invoked across the Web. They perform functions that can be anything from simple requests to complex business processes involving multiple simple services. Once a Web Service is deployed, other applications can discover and invoke the service. At present, Web Services require human interaction for identification and implementation.

Tim Berners-Lee has suggested that the integration of Web Services and the Semantic Web could offer significant performance improvement for Web applications. Integration could combine the business logic of Web Services with the Semantic Web's meaningful content. There are several areas where the two could work well together. For example, the current technologies for discovery (Universal Description, Discovery and Integration, UDDI), binding (Web Services
Description Language, WSDL), and messaging (Simple Object Access Protocol, SOAP) technologies could use OWL to provide an ontology for automatic Semantic Web Services thereby allowing interaction with Web business rules' engines.

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 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 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 SEARCH

Semantic search methods can augment and improve traditional search results by using, not just words, but concepts and logical relationships. There are two approaches to improving

search results through semantic methods: (*1*) the direct use of Semantic Web metadata and (*2*) Latent Semantic Indexing (LSI).

The Semantic Web will provide more meaningful metadata about content, through the use of RDF and OWL documents that will help to form the Web into a semantic network. In a semantic network, the meaning of content is better represented and logical connections are formed between related information.

However, most semantic-based search engines suffer increasingly difficult performance problems because of the large and rapidly growing scale of the Web. In order for semantic search to be effective in finding responsive results, the network must contain a great deal of relevant information. At the same time, a large network creates difficulties in processing the many possible paths to a relevant solution. We once again find ourselves facing a basic trade-off between finding the minimum necessary expressive power and the maximum possible reasoning capability for the Semantic Web.

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 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.

SEMANTIC BIOINFORMATICS

The Semantic Web could unlock a great deal of scientific data contained within disparate applications' formats otherwise limited by institutional factors. Life scientists, in particular, could find the Semantic Web a useful tool. The World Wide Web Consortium recently announced the formation of the Semantic Web Health Care and Life Sciences Interest Group (HCLSIG) aimed to help life scientists tap the potential benefits of using Semantic Web technology by developing use cases and applying standard Semantic Web specifications to healthcare and life sciences problems.

The initial foundation and early growth of theWeb was based in great part on its adoption by the high-energy physics community when six high-energy physics Web sites collaborated allowing their participating physicists to interact on this new network of networks. A similar critical mass in life sciences could occur if a half dozen ontologies for drug discovery were to become available on the Semantic Web.

Life science is a particularly suitable field for pioneering the Semantic Web. For example, in the area of drug discovery, many databases and information systems are used by drug researchers on a global scale. In this regard, the Biological Pathways Exchange (http://www.biopax.org/) is developing a standard data exchange format for metabolic, signaling, genetic regulatory, and genetic pathway information as an example.

KNOWLEDGE BASE

In a number of parallel efforts, knowledge systems are being developed to provide semantic-based and context-aware systems for the acquisition, organization, processing, sharing and use of the knowledge embedded in multimedia content. Ongoing research aims to maximize automation of the complete knowledge lifecycle and to achieve semantic interoperability between Web resources and services.

In one particularly interesting application, Cycorp intends to sell products and services using its inference engine, which has been designed to work with the Cyc Knowledge. Cycorp provides a reference Cyc Server executable for Intel-based Linux and for Windows 2000.

OpenCyc is the open source version of the Cyc technology, the world's largest and most complete general knowledge base and common sense reasoning engine. Cycorp, the builders of Cyc, have set up an independent organization Open- Cyc.org, to disseminate and administer OpenCyc. OpenCyc can be used as the basis for a wide variety of intelligent applications, such as speech understanding (using the KB to prune implausible choices via common sense, discourse context, and prosodics), database integration and consistency-checking, rapid development of an ontology, and email prioritizing, routing, summarizing, and annotating.

XML-BASED WEB SERVICES

Web Services provide a standard means of interoperating between different software applications running on a variety of platforms. The XML provides the extensibility and language neutrality that is the key for standard-based interoperability of Web Services.

Web Service discovery and composition is led by Universal Description Discovery and Integration (UDDI) developed by IBM and Microsoft. Well accepted standards like Web Services Description Language (WSDL) for binding and Simple Object Access Protocol (SOAP) for messaging make it possible to dynamically invoke Web services.

Web Service Architecture requires discrete software agents that must work together to implement functionality. Furthermore, the agents do not all operate in the same processing environment so they must communicate by protocol stacks that are less reliable than direct code invocation. This requires developers to consider the unpredictable latency of remote access, and take into account issues of partial failure and concurrency.

In XML-based Web Services, an agent sends and receives messages based upon their architectural roles. If a requester wishes to make use of a provider's Web Service, he uses a requester agent to exchange messages with the provider agent. In order for this message exchange to be successful, the requester and the provider must first agree on both the semantics and the mechanics of the message exchange.

The message exchange mechanics are documented using WSDL. The service description is a specification that can be processed by a machine using message formats, data types, and protocols that are exchanged between the requester and provider. It also specifies the network location of the provider.

CREATING AN OWL-S ONTOLOGY FOR WEB SERVICES

Creating an OWL-S based Ontology for a Web Service requires five steps:

1. Describe individual programs: describe the individual programs that comprise the service. The process model provides a declarative description of a program's properties.
2. Describe the grounding for each atomic process: relate each atomic process to its grounding.
3. Describe compositions of the atomic processes: describe the composite process that is a composition of its atomic processes.
4. Describe a simple process: describe a simple process for the service (optional).
5. Profile description: provide a declarative advertisement for the service. It is partially populated by the process model.

SEMANTIC SEARCH TECHNOLOGY

As Web ontology becomes more advanced, using RDF and OWL tags will offer semantic opportunities for search.

Searching Techniques

Semantic search deals with concepts and logical relationships. If we examine the practical problems of semantic search, we will find that the search tree faces an incompleteness of logic resulting in the Incompleteness Problem, or the Halting Problem.

Inference can be viewed as a sequence of logical deductions chained together. At each point along the way, there might be different ways to reach a new deduction. So, in effect, there is a branching set of possibilities for how to reach a correct solution. This branching set can spread out in novel ways. For example, you might want to try to determine "Whom does Kevin Bacon know?" based on information about his family relationships, his movies, or his

business contacts. So, there is more than one path to some conclusions. This results in a branching set of possibilities. Therefore, the inference in our system is a kind of search problem, displayed as a search tree.

It is possible to start at the top of the tree, the root, or with the branches. Taking the top of the tree, the query can be asked, Whom does Kevin Bacon know? Each step down from parent-to-child nodes in this tree can be viewed as one potential logical deduction that moves toward trying to assess the original query using this logical deductive step. The fan out of possibilities can be viewed as a branching tree, getting bushier and deeper. Each of the successful steps we take ends up becoming a parent node from which we seek additional child nodes. Eventually, a list of people "whom Kevin Bacon actually knows?" will be accumulated.

Imagine that each node in this tree represents a statement or fact to prove. Each link from a parent node to a child node represents one logical statement. Now the problem is that we have a big tree of possibilities and this could result in any search being limited to incomplete results.

In a complex logical system, there is an arbitrarily large number of potential proofs. Some of the potentially factual nodes may be arbitrarily long, and it may be uncertain if a determination of whether or not it is factual can be made (i.e., it may be uncertain if there is a proof). G¨odel proved in the 1930s that any sufficiently complicated logical system is inherently incomplete (Undecidable). In other words, there are statements that cannot be logically proven. His argument in proving undecidability is also related to the Halting Problem.

The Halting Problem is a decision problem that can be informally stated as follows: Given a description of an algorithm and a description of its initial arguments, determine whether the algorithm, when executed with these arguments, ever halts (the alternative is that it runs forever without halting). Alan Turing proved in 1936 that there is no general method or algorithm that can solve the halting problem for all possible inputs.

The importance of the Halting Problem lies in the fact that it was the first problem to be proved undecidable. Subsequently, many other such problems have been described; the typical method of proving a problem to be undecidable is to reduce it to the Halting Problem.

The Halting Problem implies that certain algorithms will never end in a definite answer. When you consider the Web, you referring to millions of facts and tens of thousands of rules that can chain together in arbitrarily complicated and interesting ways; so the space of potential proofs is infinite and the tree becomes logically infinite. Due to this, you will run into some inherent incompleteness issues; for example, in a complex network, you cannot simply look at every possible factual statement, determine its truthfulness, and collect a complete set of all such results.

You run into incompleteness because the search tree is too large. So our approach must be to search only portions of the tree. There are well-known strategies for how one addresses search problems like this. One strategy is to search the tree in a depth-first fashion.

A depth-first search would start at the top of the tree and go as deeply as possible down some path, expanding nodes as you go, until you find a dead end. A dead end is either a goal (success) or a node where you are unable to produce new children. So the system cannot prove anything beyond that point.

Let us walk through a depth-first search and traverse the tree. Start at the top node and go as deeply as possible:

1. Start at the highest node.

2. Go as deeply as possible down one path.
3. When you run into a dead-end (i.e., a false statement), back-up to the last node that you turned away from. If there is a path there that you have not tried, go down it. Follow this option until you reach a dead-end or a goal (a true statement with no child nodes).
4. If this path leads to another dead-end, go back up a node and try the other branches.
5. This path leads to a goal. In other words, this final node is a positive result to the query. So you have one answer. Keep searching for other answers by going up a couple more nodes and then down a path you have not tried.
6. Continue until you reach more dead-ends and have exhausted search possibilities.

The advantage of depth-first search is that it is a very algorithmically efficient way to search trees in one format. It limits the amount of space that you have to keep for remembering the things you have not looked at yet. All you have to remember is the path back up. The disadvantage with depth-first search is that once you get started down some path, you have to trace it all the way to the end.

Another strategy for searching is a breadth-first search. Here you search layer by layer. First, you try to do all of the zero-step proofs, then you try to do all of the one-step proofs, and so on. The advantage of breadth-first search is that you are guaranteed to get the simplest proofs before you get anything that is strictly more complicated. This is referred to as the Ockham's Razor benefit. If there is an *n*-step proof, you will find it before you look at any $n + 1$-step proofs. The disadvantage of breadth-first search becomes apparent when you encounter huge deep trees. We also have huge bushy trees where you could have thousands, or tens of thousands, of child nodes. Another disadvantage of breadth-first searching is the amount of space you have to use to store what you have not examined as yet. So, if the third layer is explosively large, you would have to store all of the third level results before you could even look at them. With a breadth-first search, the deeper you go into the tree, the more space you will need. So, you find that each of the two traditional algorithms for search, depth-first and breadth-first, are going to run into problems with large systems.

There are two basic classes of search algorithms used to attempt to overcome the incompleteness and halting limitations: uninformed and informed. Uninformed, or blind, searches are those that have no information about the number of steps or the path cost from the current state to the goal. These searches include: depth-first, breadth-first, uniform-cost, depth-limiting and iterative deepening search. Informed, or heuristic, searches are those that have information about the goal; this information is usually either an estimated path cost to it or estimated number of steps away from it. This information is known as the search agent heuristic. It allows informed searches to perform better than the blind searches and makes them behave in an almost "rational" manner. These searches include best-first, hill-climbing, beam, A*, and IDA* (iterative deepening A*) searches. These methods can provide significant improve in search.

WEB SEARCH AGENTS

While Web search engines are powerful and important to the future of the Web, there is another form of search that is also critical: Web search agents. A Web search agent will not perform like a commercial search engine. Search engines use database lookups from a knowledge base.

In the case of the Web search agent, the Web itself is searched and the computer provides the interface with the user. The agent's percepts are documents connected through the Web utilizing HTTP. The agent's actions are to determine if its goal of seeking a Web site

containing a specified target (e.g., keyword or phrase), has been met and if not, find other locations to visit. It gets on the environment using output methods to update the user on the status of the search
or the end results.

What makes the agent intelligent is its ability to make a rational decision when given a choice. In other words, given a goal, it will make decisions to follow the course of actions that would lead it to that goal in a timely manner.

An agent can usually generate all of the possible outcomes of an event, but then it will need to search through those outcomes to find the desired goal and execute the path (sequence of steps) starting at the initial or current state, to get to the desired goal state. In the case of the intelligent Web search agent, it will need to utilize a search to navigate through the Web to reach its goal. Building an intelligent Web search agent requires mechanisms for multiple and combinational keyword searches, exclusion handling, and the ability to self-seed when it exhausts a search space. Given a target, the Web search agent should proceed to look for it through as many paths as are necessary. This agent will be keyword based. The method advocated is to start from a seed location (user provided) and find all other locations linked in a tree fashion to the root (seed location) that contains the target.

The search agent needs to know the target (i.e., keyword or phrase), where to start, how many iterations of the target to find how long to look (time constraint), and what methods should determine criteria for choosing paths (search methods). These issues are addressed in the software.

Implementation requires some knowledge of general programming, working with sockets, the HTTP, HTML, sorting, and searches. There are many languages with Web-based utilities, advanced application programming interfaces (APIs), and superior text parsing capabilities that can be used to write a Web search agent. Using a more advanced, efficient sorting algorithm will help improve the performance of the Web search agent.

The Web search agent design consists of four main phases: initialization, perception, action, and effect. In the initialization phase, the Web search agent should set up all variables, structures, and arrays. It should also get the base information it will need to conduct the hunt for the target, the goal, a place to start, and the method of searching. The perception phase is centered on using the knowledge provided to contact a site and retrieve the information from that
location. It should identify if the target is present and should identify paths to other Universal Resource Locator (URL) locations. The action phase takes all of the information that the system knows and determines if the goal has been met (the target has been found and the hunt is over).

If the hunt is still active it must make the decision on where to go next. This is the intelligence of the agent, and the method of search dictates how "smart" the Web agent will be. If a match is found, the hunt is complete, and it provides output to the user. The Web search agent moves from the initialize phase to a loop consisting of the perception, action, and effect phases until the goal is achieved or cannot be achieved.

SEMANTIC METHODS

Semantic search methods augment and improve traditional search results by using not just words, but meaningful concepts. Several major companies are seriously addressing the issue of semantic search. There are two approaches to improving search results through semantic methods: (*1*) LSI and (*2*) Semantic Web documents

MODULE V
SOCIAL NETWORK ANALYSIS AND SEMANTIC WEB

How do we rank pages on the Web? How does HIV spread? How do we explain the success or failure of entrepreneurs in terms of their business contacts? What is the advantage of terrorist networks built from loosely coupled cells?

All these questions have in common that they can be rephrased using the vocabulary of network analysis, a branch of sociology and mathematics that is increasingly applied also to questions outside the social domain. In the following we give an introduction to the main theory and methods of Social Network Analysis, which will be applied later to the analysis of social networks (Section 8) as well as semantic networks (Section 9). By no means do we expect to provide a complete coverage of any topic involved. For a more encyclopedic treatment of network analysis we refer the reader to the social network analysis reference of Wasserman and Faust [WFIG94].

While Social Science is often looked upon by researchers from the exact sciences as vague and thus necessarily inconclusive, network analysis should appeal to all as one of the most formalized branches of Social Science. Most of these formalisms are based on the simple nodes and edges representations of social networks to which a large array of measures and statistics can be applied. While some of the more sophisticated of these methods require a deep mathematical understanding to be applied correctly, the simple concepts discussed in this Chapter should be easily understood by anyone with an advanced level of secondary-school mathematics.

Social Network analysis

Social Network Analysis (SNA) is the study of social relations among a set of actors. The key difference between network analysis and other approaches to social science is the focus on relationships between actors rather than the attributes of individual actors. Network analysis takes a global view on social structures based on the belief that types and patterns of relationships emerge from individual connectivity and that the presence (or absence) of such types and patterns have substantial effects on the network and its constituents. In particular, the network structure provides opportunities and imposes constraints on the individual actors by determining the transfer or flow of resources (material or immaterial) across the network.

The focus on relationships as opposed to actors can be easily understood by an example. When trying to predict the performance of individuals in a scientific community by some measure (say, number of publications), a traditional social science approach would dictate to look at the attributes of the researchers such as the amount of grants they attract, their age, the size of the team they belong to etc. A statistical analysis would then proceed by trying to relate these attributes to the outcome variable, i.e. the number of publications.

In the same context, a network analysis study would focus on the interdependencies within the research community. For example, one would look at the patterns of relationships that scientists have and the potential benefits or constraints such relationships may impose on their work. For example, one may hypothesize that certain kinds of relationships arranged in a certain pattern may be beneficial to performance compared to the case when that pattern is not present. The patterns of relationships may not only be used to explain individual performance but also to hypothesize their impact on the network itself (network evolution). Attributes typically play a secondary role in network studies as control variables.

SNA is thus a different approach to social phenomena and therefore requires a new set of concepts and new methods for data collection and analysis. Network analysis provides a vocabulary for describing social structures, provides formal models that capture the common properties of all (social) networks and a set of methods applicable to the analysis of networks in general. The concepts and methods of network analysis are grounded in a formal description of networks as graphs. Methods of analysis primarily originate from graph theory as these are applied to the graph representation of social network data. (Network analysis also applies statistical and probabilistic methods and to a lesser extent algebraic techniques.)

It is interesting to note that the formalization of network analysis has brought much of the same advantages that the formalization of knowledge on the Web (the SemanticWeb) is expected to bring to many application domains. Previously vaguely defined concepts such as *social role* or *social group* could now be defined on a formal model of networks, allowing to carry out more precise discussions in the literature and to compare results across studies.

The methods of data collection in network analysis are aimed at collecting relational data in a reliable manner. Data collection is typically carried out using standard questionnaires and observation techniques that aim to ensure the correctness and completeness of network data. Often records of social interaction (publication databases, meeting notes, newspaper articles, documents and databases of different sorts) are used to build a model of social networks.

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 be 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

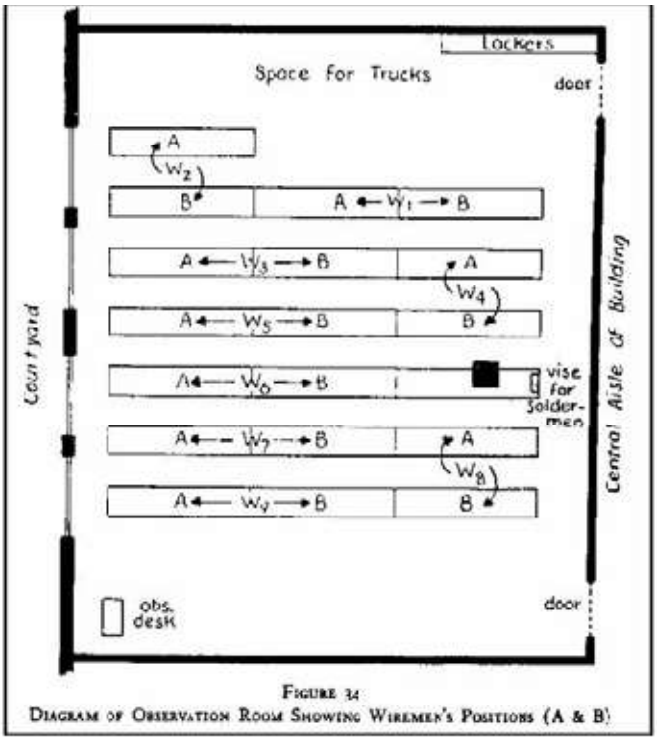

FIGURE 34
DIAGRAM OF OBSERVATION ROOM SHOWING WIREMEN'S POSITIONS (A & B)

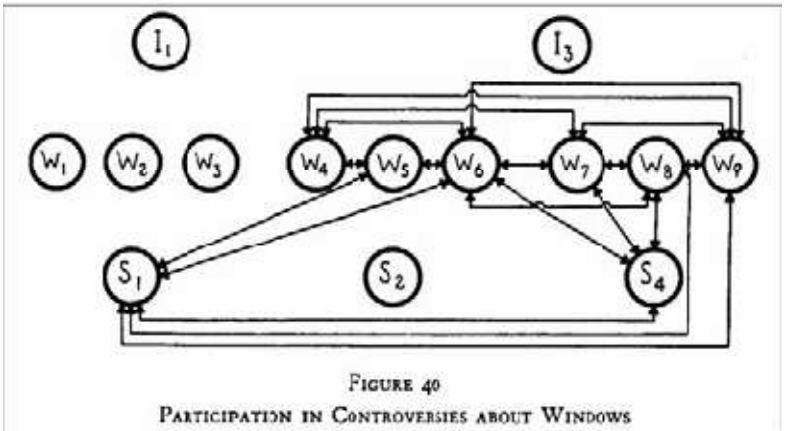

FIGURE 40
PARTICIPATION IN CONTROVERSIES ABOUT WINDOWS

Figure : Illustrations from an early social network study at the Hawthorne works of Western 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), soldiers (S) and inspectors (I) between any kind of social unit such as groups, organizations,

nations etc. While 2D and 3D visual modeling 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. 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.

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 Semantic Web community.

There are limits of course to the potential of *e-social science*. Most trivially, what is not on the Web cannot 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.

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 Huber man 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.

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.

Blogs and online communities

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 easily commenting and reacting to the comments of other 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 meetings. Figure shows some of the features of blogs that have been used in various studies to establish the networks of bloggers.

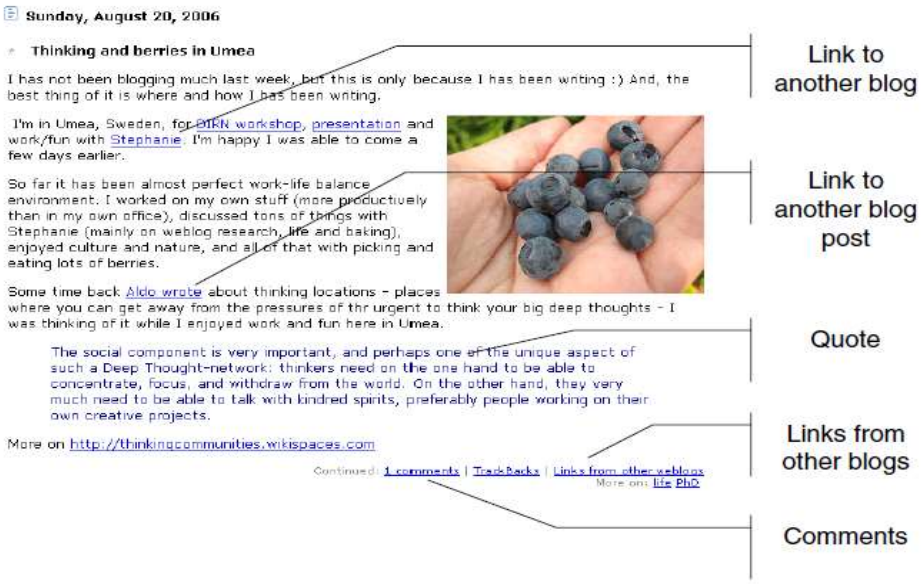

Figure: Features of blogs that can be used for social network extraction. Note also that unlike web pages in general blog entries are time stamped, which allows studying network dynamics, e.g. the spread of information in online communities.

Blogs make a particularly appealing research target due to the availability of structured electronic data in the form of RSS 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.

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. Blog analysis has suddenly shed its image as relevant only to 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, Live Journal cater to socialization even more directly than blogs with features such as social networking (maintaining lists of friends, joining groups), messaging and photo sharing. 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 Live Journal community based on the electronic data that the website exposes about the interests and social networks of its users. Backstrom et al. also study the Live Journal 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. 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.

Live Journal 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. 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

Web-based networks

The content of Web pages is the most inexhaustible source of information for social network analysis. This content is not only vast, diverse and free to access but also in many cases more up to date than any specialized database. On the downside, the quality of information varies significantly and reusing it for network analysis poses significant technical challenges. Further, while web content is freely accessible in principle, in practice web mining requires efficient search that at the moment only commercial search engines provide.

There are two features of web pages that are considered as the basis of extracting social relations: links and co-occurrences. The linking structure of the Web is considered as proxy for real world relationships as links are chosen by the author of the page and connect to other information sources that are considered authoritative and relevant enough to be mentioned. The biggest drawback of this approach is that such direct links between personal pages are very sparse: due to the increasing size of the Web searching has taken over browsing as the primary mode of navigation on the Web. As a result, most individuals put little effort in creating new links and updating link targets or have given up linking to other personal pages altogether.

For this reason most social studies based on the linking structure of the web are looking at relationships at higher levels of aggregation. For example, members of the EICSTES project investigate the web connectivity of entire scientific institutions for the purposes of *web metrics* or web-based scientometrics in the EICSTES project. For example, Heimeriks, H¨orlesberger and Van den Besselaar compare communication and collaboration networks across different fields of research using a multi-layered approach. The data for this analysis comes from bibliographic records, project databases and hyperlink networks. The connections for the latter are collected by crawling the websites of the institutions involved. In principle it could be possible to extract more fine-grained networks from the homepages of the individual researchers. However, links between homepages are too sparse to be analyzed on their own and automating this task would also require solving what is known as the home page search problem: locating the homepage of individuals given their name and description.

Co-occurrences of names in web pages can also be taken as evidence of relationships and are a more frequent phenomenon. On the other hand, extracting relationships based on co-occurrence of the names of individuals or institutions requires web mining as names are typically embedded in the natural text of web pages. (Web mining is the application of text

mining to the content of web pages.) The techniques employed here are statistical methods possibly combined with an analysis of the contents of web pages.

Web mining has been first tested for social network extraction from the Web in the work of Kautz el al. on the Referral Web project in the mid-90s. The goal of Kautz et al. was not to perform sociological experiments but to build a tool for automating what he calls *referral chaining*: looking for experts with a given expertise, who are close to the user of the system, i.e. experts who can be accessed through a chain of referrals. An example of a question that could be asked to the system is "show me all experts on simulated annealing who are at most three steps away from me in the network."

As the authors were researchers themselves, they were primarily interested in solving the referral chaining problem in the scientific domain where finding experts on a given topic is a common problem in organizing peer-reviews. The Referral Web system was bootstrapped with the names of famous AI researchers. The system extracted connections between them through co-occurrence analysis. Using the search engine AltaVista the system collected page counts for the individual names as well as the number of pages where the names co-occurred. Note that this corresponds to a very shallow parsing of the web page as indirect references are not counted this way (e.g. the term "the president of the United States" will not be associated with George Bush even if he was mentioned as the president elsewhere in the text.)

Tie strength was calculated by dividing the number of co-occurrences with the number of pages returned for the two names individually. Also known as the *Jaccard-coefficient*, this is basically the ratio of the sizes of two sets: the intersection of the sets of pages and their union. The resulting value of tie strength is a number between zero (no co-occurrences) and one (no separate mentioning, only co-occurrences). If this number has exceeded a certain fixed threshold it was taken as evidence for the existence of a tie.

Although Kautz makes no mention of it we can assume that he also filtered ties also based on support, i.e. the number of pages that can be found for the given individuals or combination of individuals. The reason is that the Jaccard-coefficient is a

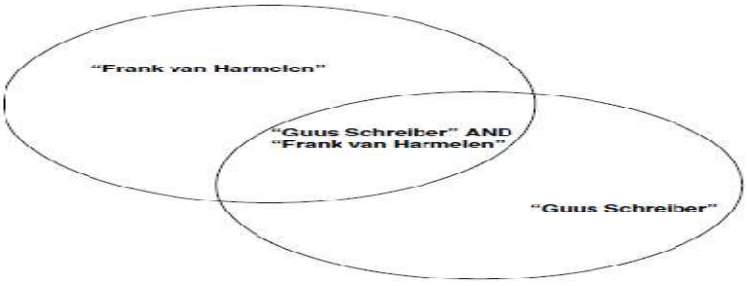

Figure: The Jaccard-coefficient is the ratio of the intersection and the union of two sets. In the case of co-occurrence analysis the two sets contain the pages where the individual names occur. The intersection is formed by the pages where both names appear.

Relative measure of co-occurrence and it does not take into account the absolute sizes of the sets. In case the absolute sizes are very low we can easily get spurious results: for example, if two names only occur once on the Web and they occur on the same page, their co-efficient will be one. However, in this case the absolute sizes are too low to take this as an evidence for a tie.

The expertise of individuals was extracted by looking for capitalized phrases that appeared in documents returned by the search engine that were not proper names. The

network in the system has grown two ways. Firstly, the documents from the Web were searched for new names using *proper name extraction*, a fairly reliable NLP technique. These names were then used to extract new names, a process that was repeated two or three times. Second, users of the system were also allowed to register themselves.

Kautz never evaluated his system in the sense of asking whether the networks he extracted are an accurate reflection of real world networks. He notes that the system as a recommender system performed well on both the research domain and in the corporate setting, although "the recommendations made by recommender system tend to be either astonishingly accurate or absolutely ridiculous true for any AI-complete problem". However, he suggest that the system is able to keep the trust of the user provided that it is made transparent. For example, the system can show the evidence on which the recommendation is based and indicate the level of confidence in its decisions. With respect to the corporate setting Kautz also notes that the results in principle can be better than using the official corporate records for locating experts as personal pages are often more up-to-date. In the scientific setting such records are non-existent and even if there existed a central system where experts can describe their social networks and expertise it would be just as likely to become obsolete on the long term as corporate yellow pages are.

In our work we use the basic method of Kautz in a slightly different way. Since our goal is the extraction of social networks we are given a list of names to begin with. We consult the search engine for investigating the possible tie between all pairs of names. Note that the number of queries required grows quadratically with the number of names, which is not only costly in terms of time but is limited by the number of queries that search engines allow. While this is not a problem in our case study, optimizations are required for larger scale analysis. A solution is proposed by Matsuo et al. who recreate the original method of Kautz by first extracting possible contacts from the results returned by the search engine for the individual names. This significantly reduces the number of queries that need to be made to the search engine at a minimal loss.

We also experiment with different measures of co-occurrence. A disadvantage of the Jaccard-coefficient is that it penalizes ties between an individual whose name often occurs on the Web and less popular individuals. In the science domain this makes it hard to detect, for example, the ties between famous professors and their PhD students. In this case while the name of the professor is likely to occur on a large percentage of the pages of where the name of the PhD student occurs but not vice versa. For this reason we use an asymmetric variant of the coefficient. In particular, we divide the number of pages for the individual with the number of pages for both names and take it as evidence of a directed tie if this number reaches a certain threshold.

Figure: The Jaccard-coefficient does not show a correlation in cases where there is a significant difference in the sizes of the two sets such as in the case of a student and a professor.

Second, we associate researchers with topics in a slightly different way. In our study of the Semantic Web community, the task is to associate scientists with research topics that have been collected manually from the proceedings of ISWC conference series. The system calculates the strength of association between the name of a given person and a certain topic. This strength is determined by taking the number of the pages where the name of an interest and the name of a person co-occur divided by the total number of pages about the person. We assign the expertise to an individual if this value is at least one standard deviation higher than the mean of the values obtained for the same concept. We also borrow from the work of Mustache and Quan Haase, who perform network analysis based on bibliographic records that contain keywords of publications. Before applying an analysis of the social-cognitive network of co-authors, the authors cluster keywords into themes based on the cooccurrences of keywords on publications, assign documents to themes and subsequently determine which themes are relevant for a person based on his or her publications. We also perform a simple clustering of keywords based on their co-occurrences among the interests of researchers.

Kautz already notes that the biggest technical challenge in social network mining is the disambiguation of person names. Persons names exhibit the same problems of polysemy and synonymy that we have seen in the general case of web search. Queries for researchers who commonly use different variations of their name or whose names contain international characters may return only a partial set of all relevant documents known to the search engine. Queries for persons with common names such as *Martin Frank* or *Li Ding* return pages about all persons with the same name. Another problem is that the coverage of the Web can be very skewed: for example, *George Bush* the president is over-represented compared to *George Bush* the beer brewer. Not only statistical methods suffer, but also content analysis as in this case the top pages returned by the search engine may not even mention the beer brewer. This is a typical web scale problem: such name collisions are rare in even the largest of corporate settings but a common phenomenon on the Web.

There have been several approaches to deal with name ambiguity. Bekkerman and McCallum deal with this problem by using limited background knowledge: instead of a single name they assume to have a list of names related to each other. They disambiguate the appearances by clustering the combined results returned by the search engine for the individual names. The clustering can be based on various networks between the returned WebPages, e.g. based on hyperlinks between the pages, common links or similarity in content. Bollegala, Matsuo and Ishizuka also apply clustering based on the content similarity but go a step further in mining the resulting clusters for key phrases. The idea is that such key phrases can be added to the search query to reduce the set of results to those related to the given target individual. For example, when searching for *George Bush* the beer brewer one would add the term *beer* to the query.

In our work in extracting information about the Semantic Web community we also add a disambiguation term our queries. We use a fixed disambiguation term (*Semantic Web OR ontology*) instead of a different disambiguation term for every name. This is a safe (and even desirable) limitation of the query as we are only interested in relations in the Semantic Web context. The method of Bollegala et al. would likely suggest more specific key phrases for every individual and that would increase the precision of our queries, but likely result in much lower recall.

We also experiment with a second method based on the concept of *average precision*. When computing the weight of a directed link between two persons we consider an ordered list of pages for the first person and a set of pages for the second (the relevant set) as shown

in Figure above. In practice, we ask the search engine for the top N pages for both persons but in the case of the second person the order is irrelevant for the computation. Let's define $rel(n)$ as the relevance at position n, where $rel(n)$ is 1 if the document at position n is the relevant set and zero otherwise ($1 \leq n \leq N$). Let $P(n)$ denote the precision at position n:

The average precision method is more sophisticated in that it takes into account the order in which the search engine returns document for a person: it assumes that names of other persons that occur closer to the top of the list represent more important contacts than names that occur in pages at the bottom of the list. The method is also more scalable as it requires only to download the list of top ranking pages once or each author. The drawback of this method is that most search engines limit the number of pages returned to at most a thousand. In case a person and his contacts have significantly more pages than that it is likely that some of the pages for some the alters will not occur among the top ranking pages.

Lastly, we would note that one may reasonably argue against the above methods on the basis that a single link or co-occurrence is hardly evidence for any relationship. In fact, not all links are equally important nor every co-occurrence is intended. For example, it may very well happen that two names co-occur on a web page without much meaning to it. What is important to realize about these methods is that they are statistical and assume that the effects of uneven weights and spurious occurrences disappear by means of large numbers.

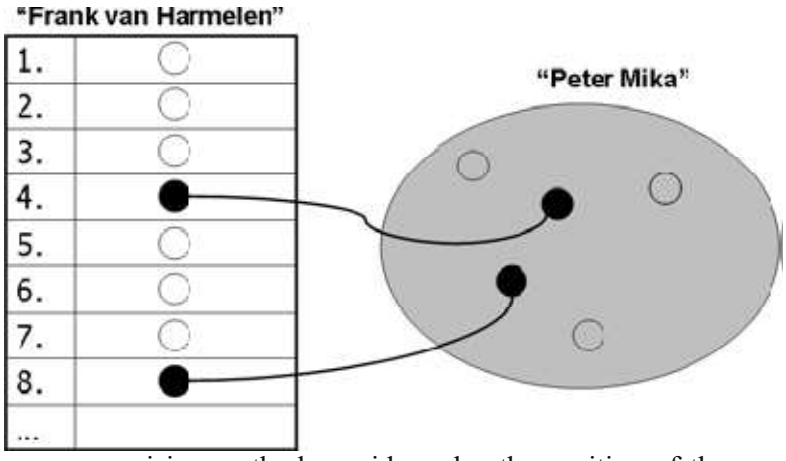

Figure: The average precision method considers also the position of the pages related to a second person in the list of results for the first person.

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 Sesame, a general database for the storing and querying RDF data. Along with and the commercial offerings of, Sesame is one of the most popular triple stores among developers, appreciated in particular for its performance. Sesame has been developed by Aduna, but available as open source.

Next, we describe the Elmo API, a general purpose ontology API for Sesame. Elmo allows manipulating 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 Graph Util which facilitates reading FOAF data into the graph object model of the Java Universal Network Graph (JUNG) API. Graph Util is open source and available as part of Flink.

The generic architecture of Semantic Web 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. 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 RDF syntax such as RDF/XML, while the latter means that the ontology's 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.

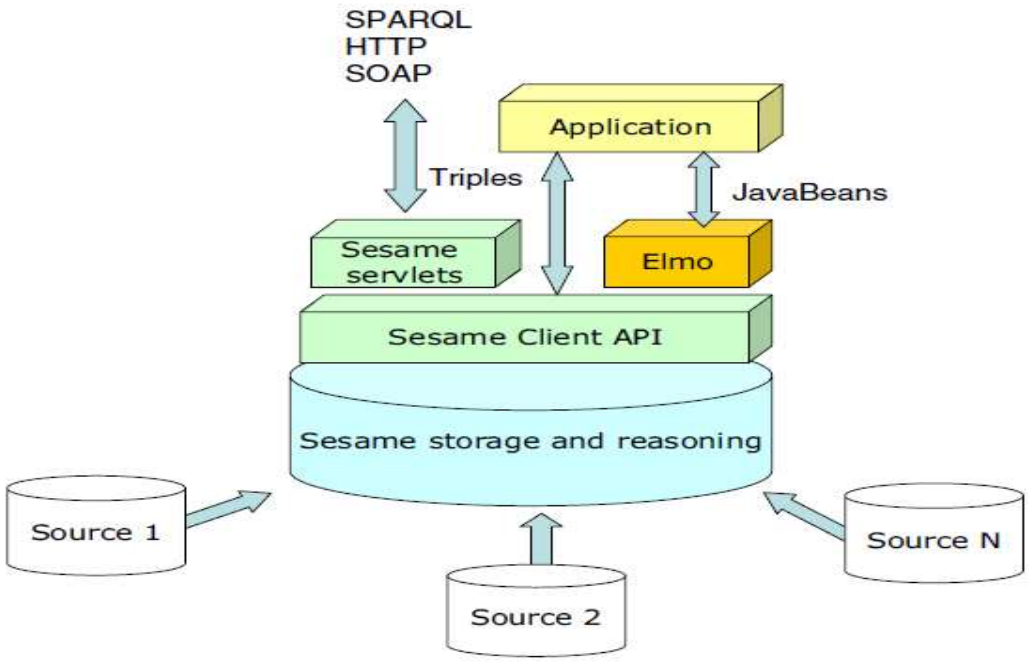

Figure: The generic design of Semantic Web applications using Sesame and Elmo. Developing with other triple stores results in similar architectures, but in general application code is not portable among triple stores due to proprietary APIs.

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 the future, it is expected that Semantic Web applications will be able to discover and select new data sources and map them automatically.

Semantic Web applications persist information in ontology stores, databases specifically designed for the storage, manipulation and querying of RDF/OWL data. Ontology stores are almost always equipped with a reasoned 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 defining custom rules that are evaluated by the reasoned 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 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 the Web in general8 In order to facilitate this, most triple stores implement the SPARQL query language and protocol 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. 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, i.e. the basic operations are adding and removing triples. Queries are given as a combination of triple patterns and return a table 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. 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, 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.

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 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 inference for applying the RDF(S) inference rules. While Sesame does not support OWL semantics, it does have a rule language that allows capturing most of the semantics of OWL, including the notion of inverse-functional properties and the semantics of the *owl*: *same As* relationship. 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 removed

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 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, 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 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 *Sesame Repository*. This object can provide access to both local Sesame servers 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, an 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.

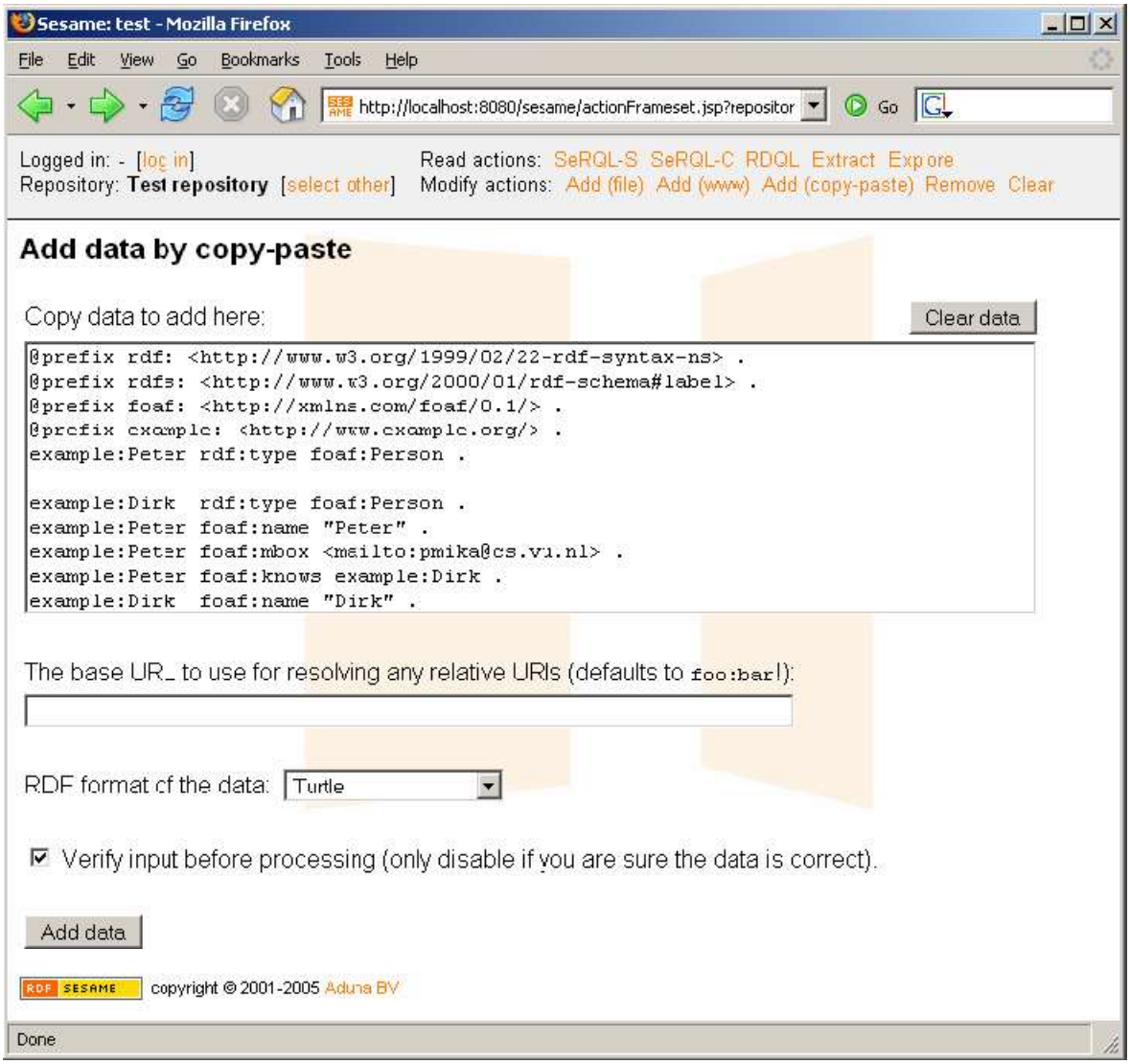

Figure: Adding data to a Sesame repository using the web interface.

There 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.

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).

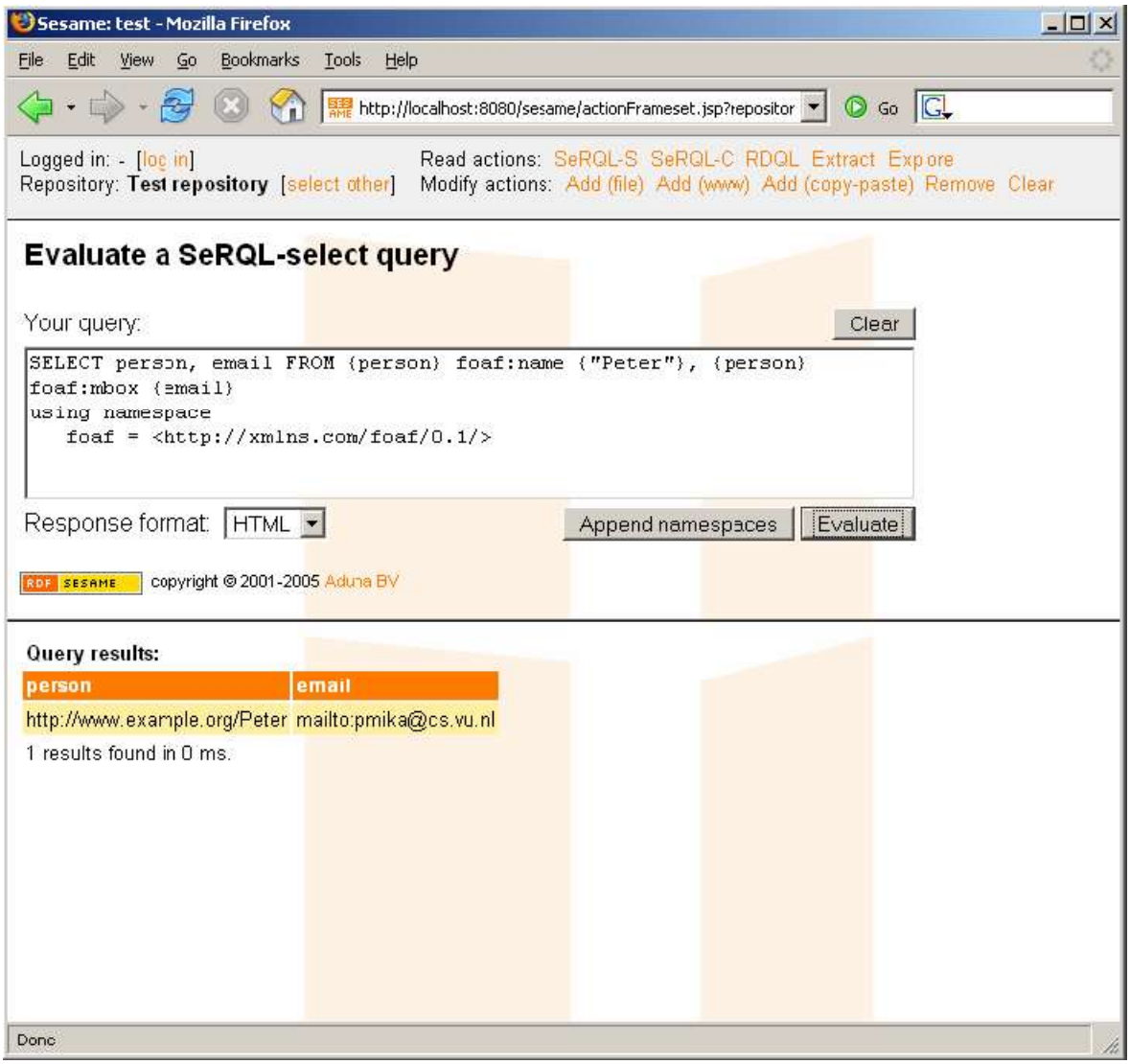

Figure : Querying data through the web interface of Sesame.

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 correspond to instances of the data set. As resources in 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 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. Behaviors 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 support 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. Creating and writing out a FOAF profile in Elmo.For example, a simple FOAF profile can be created in ten lines of Java code

```
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());
```

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. 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 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. 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. 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. 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:

• 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.

• 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 • 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 to provide a detailed logging of the crawler.

The task of the Elmo *smusher* is to find equivalent instances in large sets of data. 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 the variables are bound to actual values. 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 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 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.

GraphUtil

GraphUtil is a simple utility that facilitates reading FOAF data into the graph object model of the Java Universal Network Graph 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 that provides an object-oriented representation of different types of graphs JUNG also contains implementations for the most well known graph algorithms such as Dijkstra's shortest path. Various implementations of the *Ranker* interface allow computing various social network measures such as the different variations of centrality described. 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.

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 in Flink and open academia.

Module Wise Important Questions

Module-I

1. Discuss about the information age.
2. Discuss about the Intelligent Web Applications.
3. Discuss about the limitations of today's web.
4. Explain about the next generation web.
5. Explain about Inference Engine.
6. Explain about the Semantic road map.
7. What are the advantages of Machine Intelligence?
8. What are the advantages of Berners-Lee WWW?

Module-II

1. Discuss about the ontology's languages for the semantic web.
2. Discuss about the importance of Resource description framework.
3. Define Ontology and discuss about their role in the semantic web.
4. Discuss about the ontology web language.
5. Explain about xml scheme.
6. Explain UML.
7. Discuss in detail about OWL.
8. Write a short note on RDF scheme.

Module-III

1. Explain about various Ontology methods.
2. Discuss about the ontology development tools.
3. Discuss about the ontology sharing and merging.
4. Discuss about the ontology libraries and mapping.
5. Discuss about Logic, Rule, and Inference Engines.

6. Explain about Semantic web applications and services.
7. Discuss about XML base web services.

Module-IV

1. Explain the procedure of creating an OWL –S Ontology for web services.
2. Explain about Semantic search Technology.
3. Write a short note on Web search agents and Semantic methods.
4. Write a short note on Semantic Bioinformatics and Knowledge base.
5. Explain about OWL-S.
6. Explain about Semantic web applications and services.
7. Write a short note on Semantic search and e-Learning.

Module-V

1. What are the electronic sources for Network Analysis? Explain.
2. Discuss about electronic discussion networks.
3. Explain about Social Network Analysis.
4. Discuss about the development of Social Network Analysis.
5. Write a short note on Blogs.
6. Discuss about web based networks.
7. Explain the process of building semantic web applications with social network features.
8. Explain about online communities in detail.

CSE HOD