14 The Semantic Web – The Network of Meanings in the Network of Documents

Wernher Behrendt, Nitin Arora

Many see the Semantic Web as a logical evolution of the World Wide Web. The idea is based on the fact that nowadays, there is far too much content online on the Web for humans to find relevant information without the help of intelligent machines. The advocates for the development toward the Semantic Web led by Tim Berners-Lee identify three important supporting pillars (Berners-Lee et al. 2001). First, *semantic mark-up* - information suppliers, i.e., those who produce Web contents, will have to supply semantically marked up Web pages in the future. Second, *intelligent software agents* (that are capable of drawing inferences from the content) should be developed to search for and process such semantically marked up Web pages. And third, *computational ontologies* - the producers of Web contents and the software agents have to commit themselves to a mutually agreed understanding of things, commonly known as ontology, to make the contents also understandable for machines. According to this task sharing, we can identify three core technologies: The semantic markup uses *XML* as the carrier format and *RDF* (*Resource Description Framework*) as a first-level semantic encoding format to find and describe Web contents. The semantics of our agreed ontology is encoded within the RDF-code by use of a special (second-level) description language, the *Web Ontology Language* (*OWL*). So, our OWL-based semantic mark-up is embedded in RDF, which in turn is encoded in XML. The software agents must understand at least one ontology, can search or ask for Web contents that may likely be of interest for end-users according to the agent ontology and the search terms, lastly forming the active component of the Semantic Web. Altogether, the Semantic Web is undoubtedly still in its infancy, but many researchers and technologists from the industrial environment think that it is a promising technology for the future which will have a massive influence, particularly on the way that "knowledge workers" will use the WWW in their work, in the years to come.

14.1 Fundamentals of the Semantic Web

The term "Semantic Web" was coined at the latest in 1998 (Bernstein 1998). But the issue had been discussed in 1996 and 1997, and its basic characteristics had been described in an article

about SHOE, an HTML-based ontology description language (Luke et al. 1997). In their seminal article published in the *Scientific American*, Tim Berners-Lee, James Hendler, and Ora Lassila eventually formulated this thesis for a broader public in 2001 (Berners-Lee et al. 2001). They said that current technologies are barriers hindering the Web's evolution, and that three ingredients would help to make a quantum leap: first, the use of *software agents*, which can scan the Web for useful information on behalf of a human contractor; second, the use of new *description languages for semantic markup* of knowledge spaces for these software agents, which don't understand the human language yet, so they can communicate based on formal logics at best; and third, the creation and use of widely accepted standards for knowledge structures, systematized in *ontologies*.

14.1.1 The Role of Software Agents

The use of software agents is motivated by the wealth of information, which makes a "manual" search increasingly difficult and slow. Though we could imagine search engines that develop a more exact image of users' informational needs, there is an inherent risk of becoming a transparent society, since they require the private details of individuals to be stored in a central location. So, this approach does not represent an attractive business model from the users' perspective. In contrast, the idea of private "soft-bots", which have to account to their owners only, seems to be much more attractive, provided their right for non-violation is legally secured and supported by technology (e.g., encryption). A possible and meaningful synthesis of search engines and agent technologies could consist in that personal agents register their owners anonymously with search engines, so that both the personalization need in e-business (to collect customer needs) and the individual's privacy could be taken into account.

Wooldridge (2002) uses the term "agent" to denote a software-based computer system that has the following properties:

1. *Autonomy*: Agents operate without the direct intervention of humans, and have some kind of control over their actions and internal states.
2. *Social ability*: Agents interact with other agents and humans in some kind of agent communication language.
3. *Reactivity*: Agents perceive their environment and respond in a timely fashion to changes that occur in it. This could mean that an agent spends most of its time in a kind of sleep state from which it will awake if certain changes in its environment give rise to it.
4. *Proactivity*: Agents do not simply act in response to their environment; they are able to exhibit goal-directed behavior by taking the initiative.
5. *Temporal continuity*: Agents are continuously running processes (either active in the foreground or sleeping/passive in the background), not once-only computations or scripts that map a single input to a single output and then terminate.
6. *Goal orientedness*: Agents are capable of handling complex, high-level tasks. The agent itself should make the decision how such a task is best split up into smaller sub-tasks, and in which order and in what way these sub-tasks should be best performed.

Other desirable agent properties include *mobility* (an agent's ability to move around in a network), *veracity* (the assumption that an agent will not deliberately distribute misinformation),

benevolence (the assumption that every agent will always try to do what is asked of it), *rationality* (the assumption that an agent will always act in order to achieve its goals), and *adaptivity* (an agent's ability to adjust itself to its user's habits, working methods, and preferences). Consequently, agents have to have a considerable functionality to be able to meet these requirements. In particular, they should have certain patterns of action and communicative abilities. In addition, agents should have – limited – cognitive abilities to perceive changes in their "environment".

One of the early agent architectures called "Icarus" shows the functionalities that should interact in an agent (Langely et al. 1991). Figure 14-1 shows Icarus as an example of agent architectures.

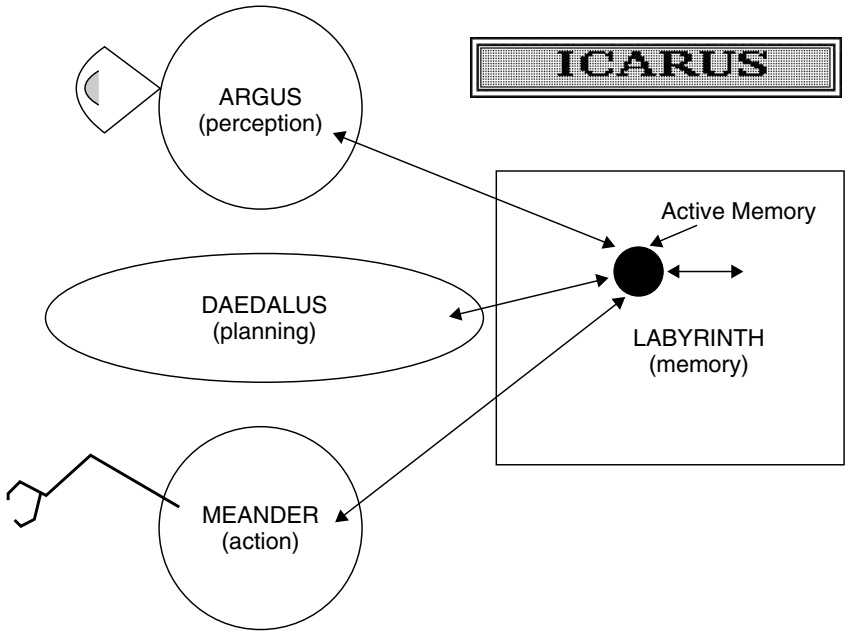


Figure 14-1 The classic "Icarus" agent architecture (1991).

We can see in this figure that *Argus*, the perceptual module, parses and transforms the environment into qualitative states, which it writes to the active memory area of *Labyrinth. Daedalus*, the planning module, is responsible for creating plans to solve problems posed by Labyrinth. *Meander*, the effector module, produces an action according to the plan constructed in active memory by Daedalus, and informs the memory of the status, while the planning module can use the memory to think about the next steps.

A methodically excellent and systematic introduction to the modeling of intelligent agents can be found in Russel and Norvig's textbook on Artificial Intelligence (Russel and Norvig 2002) (2nd Ed.).

Agent systems are actually meaningful only if there are plenty of agents, and if they are grouped in different ways, thus developing a discernable social behavior. One example is

the soccer world championship for robots, where two teams of jointly acting agents each face one another. So, it is actually a question of the "social structure", within which agents organize themselves. A classic organizational model is the so-called "Contract Net" (Smith 1981), which specifies *manager* and *contractor* roles. Agents in the Contract Net (CN) are basically built alike, but their behavior is role-specific and differs, depending on whether an agent should fulfill a task, or whether the agent itself has contracted another agent, for the fulfillment of a task. The original use scenario behind the CN involved self-regulating load distribution systems in telecommunications networks. For the Semantic Web, a role distribution suggested by Wiederhold that distinguishes between *mediators* and *facilitators* is of interest (Wiederhold 1992). Wiederhold starts from the scenario that agents have to carry out some tasks on behalf of humans, and that a rigid RPC binding would be impossible in large networks. Instead, a dynamic allocation of requests across mediators would be required, because not every agent can know everything about all other agents that could possibly process its request. Mediators are responsible for aggregating, abstracting, and integrating information to yield homogeneous information structures on top of which value-added services can be built. Facilitators are similar to directory services, knowing which services can be obtained and from where, establishing contacts between requesters and service providers. In this scenario, it is possible that a group of mediators has subscribed with a facilitator, who, in turn, is in touch with other facilitators, who manage the services. This approach allows for self-organizing work distribution in networks.

14.1.2 The Role of Semantic Markup

In order for software agents to recognize whether or not a piece of information found on the WWW is usable for a given purpose, the World Wide Web Consortium (W3C) specified that Web pages have to include a meta-data record suitable for interpretation by software in order to be useful for the Semantic Web. One of the first languages for semantic markup was SHOE (Simple HTML Ontology extension, Luke et al. 1997). DAML+OIL (DARPA Agent Markup Language with the Ontology Inference Layer) was developed later on, and led to a joint initiative by the name of "OWL" (Web Ontology Language) in 2002. Section 14.2 introduces OWL. However, the original language SHOE is still useful because its structure is simple, thus offering a better insight into what is to be achieved by semantic markup. Figure 14-2 shows how SHOE can be used for the semantic description of a university professor's Web page (`http://www.cs.umd.edu/users/hendler/sciam/step2.html`).

The example in Figure 14-2 describes an instance (Dr. Hendler, the university professor), referencing an ontology (`cs-dept-ontology`) in a structured way, so that it is possible, for instance, to express where the professor obtained his academic title (`<RELATION NAME="cs.doctoralDegreeFrom">`).

An agent that understands SHOE can now find the university where Mr. Hendler obtained his doctorate, and could perhaps communicate with an agent of that university to find out whether this information is correct.

The use of a standardized language for semantic markup is important, as we can understand from the above discussion. But the real challenge is to develop generally acceptable and binding ontologies upon which the agents' semantic understanding will be based.

```
<INSTANCE KEY="http://www.cs.umd.edu/users/hendler/">
   <USE-ONTOLOGY ID="cs-dept-ontology"
   VERSION="1.0" PREFIX="cs"
   URL="http://www.cs.umd.edu/projects/plus/SHOE/cs.html">
   <CATEGORY NAME="cs.Professor"
   FOR="http://www.cs.umd.edu/users/hendler/">
   <RELATION NAME="cs.member">
     <ARG POS=1
VALUE="http://www.cs.umd.edu/projects/plus/">
     <ARG POS=2
VALUE="http://www.cs.umd.edu/users/hendler/">
   </RELATION>
   <RELATION NAME="cs.name">
    <ARG POS=2 VALUE="Dr. James Hendler">
   </RELATION>
   <RELATION NAME="cs.doctoralDegreeFrom">
     <ARG POS=1
VALUE="http://www.cs.umd.edu/users/hendler/">
     <ARG POS=2 VALUE="http://www.brown.edu">
   </RELATION>
   <RELATION NAME="cs.emailAddress">
     <ARG POS=2 VALUE="hendler@cs.umd.edu">
   </RELATION>
   <RELATION NAME="cs.head">
     <ARG POS=1
VALUE="http://www.cs.umd.edu/projects/plus/">
     <ARG POS=2
VALUE="http://www.cs.umd.edu/users/hendler/">
   </RELATION>
</INSTANCE>
```

Figure 14-2 Using SHOE for semantic markup of a Web page.

14.1.3 The Role of Ontologies

Ontologies are conceptualizations that codify the knowledge of experts in a form that can be comprehended and reconstructed by experts. Ontologies are not necessarily subject to a formal semantics in the logical-mathematical sense. If their semantics is non-formal, their notion of truth is then only measurable as the degree of the informal consensus between the users of the ontology. If an ontology is available in the form of an axiomatized theory that enables a model-theoretic interpretation, then its semantics is formal, enabling statements about its logical truth to be checked. Dealing with ontologies has at least two roots in informatics, namely data modeling as part of database research, and knowledge representation as part of research in the field of artificial intelligence (AI). For example, databases have an explicit "ontology", namely the database schema based on a data model. However, this schema normally maps a very limited section of the real world into the computational system, because the database application orients itself to the respective problem, thus being suitable for logical deductions only within narrow boundaries. In practice, such a schema is interpreted only by the specified applications, which use the schema to access data. AI takes a broader look by studying whole application domains for which it develops

semantic networks and rule systems based on predicate logic. AI has developed different description formalisms for these domain models, including specific description languages for ontologies.

Today, ontologies serve two purposes. They are used either as "global integration schemas", for example, when building portals and data warehouses, or as controlled vocabularies for document annotations (meta-data). Only very few "true" AI applications use ontologies based on data to construct further-reaching inferences (forward or backward chaining rule systems), which can ultimately serve in decision-finding processes.

Opinions in practice differ about the necessary degree and method of systematizing ontologies. Many "ontologies" are actually no more than controlled vocabularies. Though some are built hierarchically in the form of taxonomies, the type of hierarchization can be quite varied, reaching to formally defined inheritance hierarchies in object-oriented models. Very few ontologies stand up to a philosophical-ontological analysis, in addition to the existence of formal semantics (If my dog is muzzled, is the muzzle part of the dog, and why or why not? If the dog is mine, is his muzzle also mine, and what consequences does it have?). Such questions are not obscure, because if you tell your software agent to get a dog muzzle, you'd probably want to make sure it won't buy a dog only because it can logically deduce that when you buy a dog you also get yourself a muzzle. So, those who seriously deal with ontology-based agent systems are well advised to use trustworthy knowledge models. Issues relating to ontologically clean modeling have been addressed by Guarino and Welty (2002) and others. We can see that the Semantic Web is not only a technical challenge, but also a philosophical and socio-economic one, especially if we look at the goal the European Union (EU) set itself in 2000, that, by 2010, it should become the most competitive and dynamic knowledge-based economy in the world (the goals have recently - 2005 - been toned down, but the aspiration is still there). The EU is currently funding significant research in the field of the Semantic Web and will probably continue to do so in the next three to five years.

14.2 Technological Concepts

Based on the Semantic Web architecture briefly described above, this section discusses agent standards, ontologies, and markup languages for the Semantic Web. The FIPA standard is currently the most matured for agent systems (FIPA 2002). With regard to ontologies, we have to distinguish between the semantic level and the syntactic level. On the semantic level, it is important to know what can principally be expressed with a certain knowledge representation language. For example, we cannot describe countable sets in first-order logic, because the concept of cardinality is excluded from first order logic. On the syntactic level, it is important to represent different semantic sub-levels with the help of adequate and concrete description languages. We will show that the knowledge level (OWL) and the Web resources level (RDF) and the programming interfaces (XML) can be sufficiently represented by embedding OWL in RDF and XML to yield practicable systems.

14.2.1 Agents According to the FIPA Standard

There are currently a number of academic agent projects and also a number of different development frameworks. A good source for various approaches is AgentLink, a European

research network for agent systems (AgentLink 2003). This chapter will be limited to the FIPA standard, which has found broad industrial acceptance, and which can be used to clearly represent the technical components of agent systems. The Foundation for Intelligent Physical Agents (FIPA) is a non-profit organization to promote the standardization and further development of agent systems (FIPA 2003).

Figure 14-3 shows the general model of an agent communication, assuming that an *agent*, X, follows a *goal*, G. The agent will approximate goal G by an *intention*, I. Now, intention I has to be converted into a *speech act*, S. To realize speech act S, a *message*, M, has to be sent to agent Y. This message is decomposed into transport packets similarly to a remote procedure call (RPC), and eventually converted into an intention of agent Y on the meanings level.

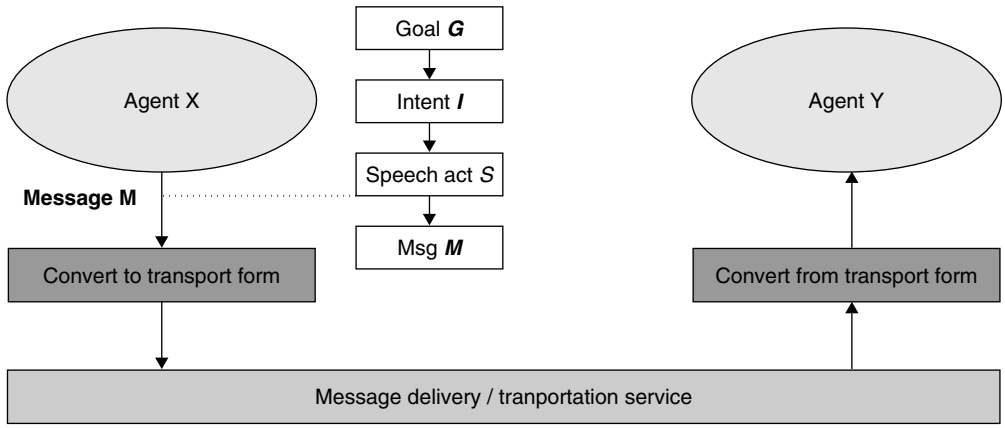


Figure 14-3 The FIPA agent communication model (Foundation for Intelligent Physical Agents 2002).

FIPA uses *ACL* (*Agent Communication Language*), which comprises five syntactic categories:

- Type of communication act (*performative*);
- Agents participating in the communication (*sender, receiver, reply to*);
- Content of message (*content*);
- Description of the content (*language, encoding, ontology*); and
- Control of conversation (*protocol, conversation identifier, reply with, in reply to, reply by*).

The only parameter that is mandatory in all ACL messages is the performative, although it is expected that most ACL messages will also contain sender, receiver, and content parameters to be able to reproduce a speech act. However, a valid minimum conversation would simply consist of a "request", followed by an "accept" or "not understood" reply.

Agents and Their Ontologies

Of course, the discussion about the Semantic Web has given the software agent issue an additional lift. Consequently, there are currently FIPA standardizations for three variants of the content

description language, namely for the Knowledge Interchange Format (KIF), the Constraint Choice Language (CCL), and the Resource Description Framework (RDF). In addition, a possibility has been created for groups of agents to commit themselves to a common ontology ("ontological commitment"), which will be made available to them by a specialized ontology agent. All agents can make requests to the ontology agent. This mechanism supports a better interplay between ontology vendors and application-specific users (agents). The Ontology Service Specification defines how agents can "load" themselves with an ontology and then act according to this ontology. Despite this promising work, no implementations that seriously use this concept are currently known.

14.2.2 Ontologies

When using the term "ontology", we have to distinguish between the philosophical discipline of ontological analysis and the product of such an analysis, i.e., the description of a specific knowledge space.

Philosophy, Ontology, and Language Criticism

Before describing the technical implementation of ontologies, one is well advised to undertake a brief excursion into ontology as a field of philosophy. This little excursion is aimed at showing the potential difficulties that could result from implementing solutions for the Semantic Web, if one tackled ontology-building in a linguistically naive manner.

Ontological analysis has long been an important part of philosophical reasoning. In the Western culture, ontology dates back to Aristotle. According to Aristotle, the subject of ontology is the study of categories of things that exist or may exist in some domain. Aristotle distinguished ten basic categories for classifying anything that may be said or predicated about anything, and devised rules for combining them to create new concepts or sub-categories. In the 18th century Kant proposed an alternative form of categorization, which will not be discussed here; interested readers are referred to (Sowa 2000). Since the middle of the 19th century, philosophers have tried to separate the *categories* of human thinking from its *linguification*. This has led to a "language-critical philosophy" which, for example, identifies very clear correlations between the grammar of Indo-Germanic languages and Aristotelian categories (Köller 1988). Now, did Aristotle find the basics of human knowledge, or only the grammar structures of the Greek language? After all, mathematical logic is also based on these categories! So, in view of working out ontologies that can be interpreted by Web applications and software agents, we must be aware that these ontologies may well be pragmatic and useful, but that all too often, they are based on a rather *naïve language understanding*. We have to understand that, although a "knowledge model" created in this way could be helpful for data exchange, it can hardly claim ontological generality. Let's look at an analogy about cars to better understand this problem: Though driving a car is faster than walking, we have to accept that cars generally work well only on roads, and that they are suitable neither for climbing stairs nor for swimming. This means that technically implemented ontologies can have a clearly defined and specific benefit, similar to cars, but, in the world of human meanings, the notion of the automobile is not suitable to make useful statements about all forms of motion (swimming, climbing stairs,

crawling, flying). These are some of the restrictions to be expected from ontologies that were built ad-hoc.

Ontologies for Computer Applications

For the Semantic Web, we use an explicit knowledge model that will serve to represent a knowledge space. According to (Gruber 1993), "an ontology is an explicit specification of a conceptualization". In this context, we will use a terminology which, though not further analyzed, is described in a formally and logically substantiated Knowledge Representation Language (KRL; see next section). An ontology for computer applications is an attempt to explicitly represent the concepts of a knowledge space and the inherent dependencies and interplays between its concepts. Such an ontology is normally defined as the generally valid schema for all possible statements within this knowledge space (also known as the UoD - universe of discourse). Concrete specifics of this knowledge space can be maintained as instances of this schema.

Formal Representation of Human Knowledge

A Knowledge Representation Language (KRL) should be defined on a descriptive level on which any specific knowledge can be represented by specializing by the primitive terms of the representation language. The simpler the KRL, the more modeling will be required to describe even simple facts. The more powerful a KRL becomes, the more complex can be its use. In either case, every KRL has to deal with the problem that human language and human understanding of a language is highly context-sensitive, subject to conditions that change constantly, while even good knowledge-based systems have only as much flexibility as was programmed "into" them (if one views the formal definition of search spaces as "programming", which is our chosen interpretation of knowledge-based programming). Our discussion of the Semantic Web assumes that the objective is to obtain explicit knowledge descriptions for the respective use purpose that can be interpreted by agents appropriately. The responsibility for this interpretation is vested in the agent and less in the ontology, provided the ontology is based on correct knowledge with regard to at least one such interpretation. We will introduce two KRL examples below.

EER Diagrams

Entity-relationship (ER) diagrams have been known as a conceptual language for the design of relational databases since the 1970s (Chen 1976). In the 1980s and 1990s, they were enhanced to support the modeling of inheritance hierarchies and aggregation relationships (*Extended Entity-Relationship Diagram – EER* Engels et al. 1992). ER and EER models force the designer to make a distinction between *entities* and *relationships* that can arise between entities. One example from the tourism industry would be the entities *guest* and *hotel*, which can have a *relationship* through the term (guest) *stays_in* (hotel). ER and EER also allow us to model quantitative restrictions in relationships. Such a restriction would be that a guest can stay overnight in a hotel only once at a time (but the guest can book several hotels for the same night). At the same time, many guests can stay overnight in one hotel (but with an upper limit). Both the differentiation of entities and relationships and the quantitative restrictions between the end points of a relationship are

characteristic properties of all data-centric KRLs, but are sometimes excluded from inference-centric KRLs because the consideration of numeric constraints can inflate the search space for the inference engine of a knowledge-based system in prohibitive ways.

Conceptual Graphs

The idea of *conceptual graphs* (CGs Sowa 1976, Lukose et al. 1997) is based on the *existential graphs* originally developed by Charles S. Peirce in the 19th century, but which Peirce never fully completed (see Øhrstrøm 1997). A CG distinguishes between concepts and conceptual relations, where concepts can be related only through conceptual relations. The most important idea for our purposes is that CGs allow us to map complex language expressions, such as type hierarchies. Moreover, it is possible to describe nested contexts. The following examples serve to better understand this concept (see Sowa 2000, pp. 476–491). The sentence "John is going to Boston by bus" uses four concepts, namely *John, Go, Boston, Bus*. These concepts are attached to three conceptual relations: Agent (*Agent*), Instrument (*Inst*), and Destination (*Dest*), as shown in Figure 14-4.

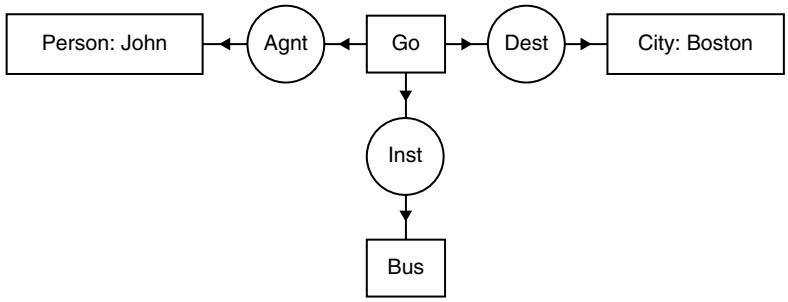


Figure 14-4 Conceptual graph for "John is going to Boston by bus".

The sentence "Tom believes that Mary wants to marry a sailor" has a nested conceptual structure, as shown in Figure 14-5.

The outer level describes the proposition that Tom believes something. Inside that context is another context of type *situation*, which describes a situation that Tom believes Mary wants. The resulting CG represents the sentence "Tom believes that Mary wants to marry a sailor". The *proposition* box has as theme (*Thme*) a situation that Mary hopes will come to pass. Inside the proposition box are three concepts: Person: Mary, Want, and the situation that Mary wants. Since those three are only asserted within the context of Tom's belief, the graph does not imply that they must exist in the real world. Since Mary is a named individual, one might give her the benefit of the doubt and assume that she exists; but her desire and the situation she supposedly desires exist in the context of Tom's belief. If his belief is false, the referents of those concepts might not exist in the real world. All of this would present a few problems for our agent: Should the agent assume that Mary really exists? And if so, does she really want to marry a sailor, or is Tom wrong and she actually wants to marry a computer scientist? We always have to be

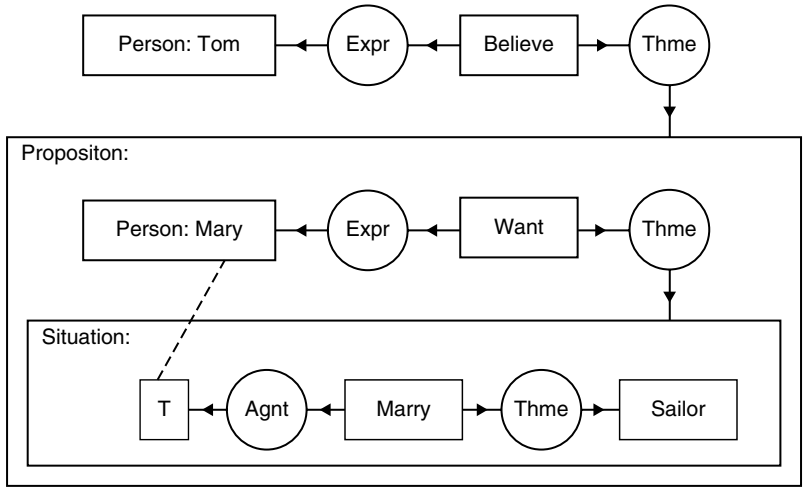


Figure 14-5 Conceptual graph for "Tom believes that Mary wants to marry a sailor".

aware that a computer system actually interprets only mathematical-logical relations, without understanding what "believe", "want", and "being sure" generally mean in human interactions. To simulate a behavior that somewhat corresponds to human behavior, both the concrete model and the underlying logic have to be very extensive. Our examples can only scratch the surface of this extensiveness.

It should be noted that there is no difference between predicate logic, the Knowledge Interchange Format (KIF), and conceptual graphs (CGs), as far as the underlying formal semantics is concerned. However, conceptual graphs use a notation that is easier to read for humans, thus giving us an insight into the requirements which an application for the Semantic Web has to meet.

14.2.3 Semantic Markup on the Web

The concepts available for semantic markup on the Web are based on the work on knowledge representation languages (KRLs; see above) during the past thirty years. The three main representatives for semantic markup are OWL, RDF, and RDF Schema.

DAML + OIL = OWL

The DAML (DARPA Agent Markup Language) description language was introduced as a KRL extension to RDF and XML in 2000 (Hendler and McGuinness 2000). Roughly at the same time, a European research project developed the Ontology Inference Layer (OIL) (Fensel et al. 2000). The protagonists of both KRLs agreed later on to continue developing a joint Web Ontology Language (OWL) (McGuinness and van Harmelen 2003).

OWL currently exists in the form of three sub-languages with increasing expressivity: *OWL-Lite* allows specifying inheritance hierarchies with associations and type restrictions, equality and

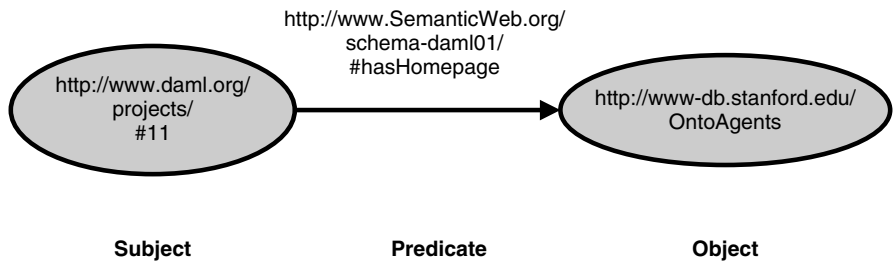
unequality relations, and transitivity and symmetry conditions for associations. One important restriction and limitation of OWL-Lite is that cardinalities can only be specified as 0 or 1 (expressing "can exist"/"must exist", respectively). The reason is said to be that systems with this restriction are easier to implement for the reason cited above, that full treatment of cardinality leads to higher computational complexity.

OWL-DL (DL stands for description logics) and *OWL-Full* enhance OWL-Lite. Though OWL-DL and OWL-Full share the same syntax, some constructs in OWL-DL have a restricted formal semantics compared with OWL-Full. For example, in OWL-DL classes cannot be used as instances due to the strict separation of instances and classes in all description languages.

For the practical use of technologies for the Semantic Web, it is important to understand how XML, RDF, RDF Schema, and OWL engage with one another, though opinions are divided about how meaningful this structure is–one might argue that a lot of new notation has been introduced for little additional semantics, compared with the KRLs that have existed for many years, such as KL-One (Brachman and Schmolze 1985), Telos (Mylopoulos et al. 1990, Nejdl et al. 2001), KIF (Genesereth 1998), and conceptual graphs (Sowa 1976, Lukose et al. 1997). In order to get a grasp of the notations and their interdependencies, the next two subsections will briefly describe how OWL is embedded in RDF and RDF Schema, using XML as the general representation format. Subsequently, we will use an example to show how these nested Web languages can be used in practice.

RDF

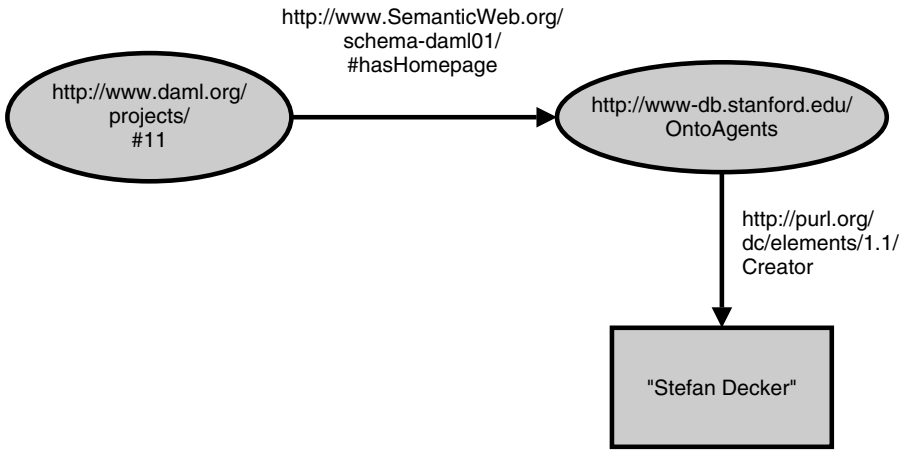
The *Resource Description Framework* (*RDF*) has been designed to connect Web-based data sources to the emerging Semantic Web at relatively low cost (Decker et al. 2000). According to its inventors, the RDF data model is similar to the object-oriented model, but we think that it is actually close to the functional data model (Shipman 1981). RDF distinguishes between entities, which are represented by unique identifiers (UIDs), and statements, which are valid between entities. This means that a statement connects a subject (source entity) and an object (destination entity) via a predicate/property. Figure 14-6 shows a schematic view of the RDF statement "The OntoAgents project has a home page at (`http://www-db.stanford.edu/OntoAgents`)". As can be seen, RDF can be thought of as using a distinction between subject, predicate, and object.



Source: http://cse.hanyang.ac.kr/~jmchoi/class-old/ 2002-2/cse995/INTRO.ppt

Figure 14-6 RDF graph with subject, predicate, and object.

The next things RDF distinguishes are *resources* and *literals*. A resource is designated by its Uniform Resource Identifier (URI), e.g., (`http://www.SemanticWeb.org/schema-daml01/#hasHomepage`). Subjects and predicates are always resources, while an object can be either a resource or a literal. A literal has no resource allocated to it in a URI; instead, it is a character string to which the predicate is allocated as a value. The schematic view shows resources as ellipses and literals as rectangles. Figure 14-7 uses the example from Figure 14-6, additionally modeling the statement ". . . and the homepage was created by Stefan Decker."



Source: http://cse.hanyang.ac.kr/~jmchoi/class-old/ 2002-2/cse995/INTRO.ppt

Figure 14-7 RDF graph with a literal allocated as a value.

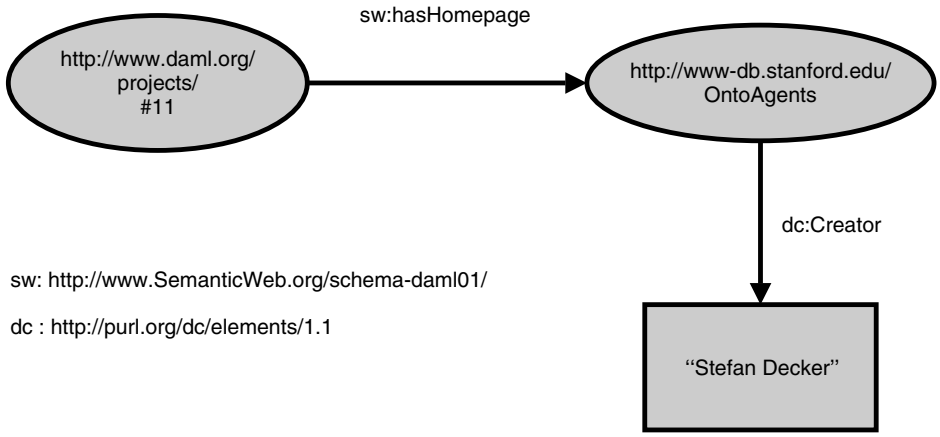
RDF is primarily designed for machine-assisted Web navigation, and therefore, it is hard to read for humans in its basic form, i.e. a series of URI-based statements. This was the reason why the notion of namespaces was subsequently borrowed from XML to improve readability. The convention is to associate a prefix with the respective URI. In our current example, such a definition is used to abbreviate (`http://www.SemanticWeb.org/schema-daml01/#`) to "sw". The result is that the predicates are shortened to `sw:hasHomepage` or `dc:Creator`, respectively (see Figure 14-8).

Figure 14-9 summarizes the entire RDF code for these statements.

RDF itself uses namespaces to define other useful constructs, which are then fitted with the `rdf` prefix, including:

- `rdf:type` : RDF predicate for type allocation.
- `rdf:Bag` : RDF predicate for an unordered set.
- `rdf:Seq` : RDF predicate for an ordered set.
- `rdf:Alt` : RDF predicate for a set of optional choices.

We can see that RDF offers a vocabulary that we can use to attach relationships between data sources on the Web. These data sources are called "resources" in RDF. The naming convention



Source: http://www.ida.liu.se/~asmpa/courses/sweb/rdf/sweb_rdf.ppt

Figure 14-8 An RDF graph using namespaces.

```
<?xml version='1.0'?>
<rdf:RDF
xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:sw ="http://www.SemanticWeb.org/schema-daml01/#"
xmlns:dc ="http://purl.org/dc/elements/1.1/">

<rdf:Description
  about="http://www.daml.org/projects/#11">
    <sw:hasHomepage>
       <rdf:Description about=
          "http://www-db.stanford.edu/OntoAgents">
          <dc:Creator>Stefan Decker</dc:Creator>
       </rdf:Description>
    </sw:hasHomepage>
  </rdf:Description>
</rdf:RDF>
```

Figure 14-9 RDF code for the home page of the OntoAgents project.

for RDF predicates (properties) does not commit us to any typing, which means that it is not really helpful yet for agent-supported applications. RDF Schema, described in the next subsection, was designed to remove this shortcoming.

RDF Schema

RDF Schema (RDF-S) uses RDF to define additional constructs useful for creating object-oriented schemas in a separate namespace. These constructs are given the prefix `rdfs`, which can

Table 14-1 RDF Schema constructs (excerpt)

RDF/RDFS Construct	Description	Comment/Example
	Class Definition	
rdfs:Resource	All things described by RDF; entities are resources.	This is the root class from which all others are derived.
rdfs:Class	Definition of a class (using rdf:type).	rdfs:Class is an instance of rdfs:Class.
rdfs:Literal		rdfs:Literal is an instance of rdfs:Class.
rdf:type	Predicate used to allocate typification.	rdf:type is an instance of rdf:Property.
rdf:XMLLiteral	Predicate used to designate XML literals.	
rdfs:subClassOf	Predicate used to specify an inheritance hierarchy.	rdfs:subClassOf is an instance of rdf:Property.

be found at (`http://www.w3.org/2000/01/rdf-schema`). Table 14-1 lists some of the important RDF Schema constructs.

If we want to express that `automobile` is a class in a user-defined schema (with `mymodel` as prefix), we form an RDF-3 tuple (*triple*):

```
mymodel:automobile
rdf:type
rdfs:Class
```

To derive a subclass, `lorry`, from `automobile`, we write the following 3-tuple:

```
mymodel:lorry
rdfs:subClassOf
mymodel:automobile
```

We can now allocate associations to the schema for `automobile`. First, we use a type assignment, `weight-in-kg`, to declare an association (attribute), and then we specify that `weight-in-kg` is assigned to the `automobile` class:

```
mymodel:weight-in-kg
rdf:type
rdf:Property

mymodel:weight-in-kg
rdfs:range
mymodel:automobile
```

OWL uses RDF Schema to form class hierarchies and to declare associations between classes. Table 14-2 lists the most important OWL constructs. RDF Schema, in turn, uses RDF constructs, like `rdf:type` or `rdf:Property`, to describe a schema.

Table 14-2 OWL constructs and their relationships to RDF and RDF-S

OWL Construct	Description	Comment/Example
owl:Class	Defines a class.	
rdf:Property	Defines an association.	Is taken from RDF as definition.
rdfs:subClassOf	Is a subclass.	Is taken from RDF Schema as definition.
rdfs:subPropertyOf	Is a sub-association.	See above – RDF Schema
rdfs:domain	Is the mathematical domain of a function.	See above – RDF Schema
rdfs:range	Is the mathematical range of a function.	See above – RDF Schema
owl:Individual	Is the declaration of an instance.	

Neither XML nor RDF are expressive enough to represent knowledge structures such as ontologies well, and to model them in a sufficiently formal manner. This was the reason why both the XML protagonists and the RDF advocates have defined languages in the form of XML Schema and RDF Schema, respectively. The expressivity of both these languages is somewhere between RDF and OWL. The layered model for Web languages aspired to by Tim Berners-Lee has not yet been achieved, because description languages overlapping the range from XML to OWL have been constructed, which, when interacting, have many different possibilities to represent the same fact. It is impossible to say today which description languages will eventually gain acceptance in the future. What *can* be said is that today's choice (2005) for Semantic Web *research* applications is the XML-RDF-OWL stack, whereas *industrial* solutions usually content themselves with application-specific XML-RDF descriptions.

14.3 Specifics of Semantic Web Applications

The technologies currently emerging for Semantic Web applications have not yet reached industrial maturity, so that we can't really speak of much industrial engineering practice in this field. Nevertheless, many innovative vendors have introduced initial components or prototypes to the market.

14.3.1 Semantic Markup

In practical applications, semantic annotations using some form of mark up will likely be the most frequent customer requirement in connection with the development of a Semantic Web application. If concepts of the Semantic Web are to establish themselves in practical applications at all, then corresponding agent systems will have to be developed, too. Semantic markup requires, firstly, a good ontological model and, secondly, the annotators' sound understanding of the application-specific ontology.

14.3.2 Agents

In practice, we will probably have to deal with the problem that, though most known semantic markup methods are relatively mature, only a few solutions introduce proper agent systems, which understand semantic markup. On the positive side, however, agent-based solutions can be replaced by more conventional solutions, for instance *Web services*. These more conventional solutions use the ontology to integrate schemas from existing, previously heterogeneous data repositories. To this end, Web services are used to act as wrappers for the data repositories. The DAML-S description language (for Web Services) tries to contribute in this field (Ankolenkar et al. 2002).

However, if the target application requires a high degree of complex and autonomous behavior, then an additional group of mediators and perhaps even facilitators have to be added to the integration layer. This type of target application is characterized by a heavily distributed multi-layer architecture, which is currently still associated with high development risks. Most developer teams will probably not have all the required competencies and, at the same time, some of the available technologies are still under development. In addition, these applications are characterized by high technical complexity.

14.3.3 Ontologies

The following six principles have been identified to be characteristic for the Semantic Web (Koivunen and Miller 2001). All the listed points require some rethinking in general and about ontologies in particular, compared with the traditional approach in AI research and the technologies developed in this field.

1. *Everything can be identified by URIs.*
   Both things from the real (physical) world and things from the digital world can be referenced. The approach is pragmatic and allows, for example, to identify a person by referencing that person's employer address and e-mail.
2. *Resources and links can be typed.*
   While humans give meaning to resources and links on the traditional Web, the Semantic Web will allow to create a machine-readable way of allocating meanings to facilitate referencing online terminologies and ontologies.
3. *Incomplete information must be tolerated.*
   In contrast to many "classic" AI systems, the so-called "closed-world assumption" cannot be maintained on the Semantic Web. So, if we draw logical conclusions from a statement, then the fact that *information is missing* from the knowledge base should not automatically result in concluding that the statement is FALSE.
4. *No claim is laid to absolute truth.*
   Also in contrast to classic systems, the Semantic Web should tolerate claims to *local* levels of truth.
5. *Evolution of information or resources is possible.*
   Just as humans and things change over time, there will also be information that was correct at a certain point in time, but which should be replaced by new information at another

point in time. This doesn't mean that the old information has to be deleted or overwritten, because it may still be valid for some statements.

6. *Minimalist design.*
   The W3C initiatives plan for the Semantic Web to develop as simple mechanisms as possible, leaving much freedom for experiments. This means that the standardization is somewhat "laissez-faire", which is stimulating, but also confusing, as we can see from the fact that various standards overlap.

14.3.4 Semantic Web Services

Today's Web was designed primarily for human interpretation and use. Nevertheless, we are seeing increased automation of Web service interoperation, primarily in B2B and e-commerce applications. Generally, such interoperation is realized through APIs that incorporate hand-coded information-extraction code to locate and extract content from the HTML syntax of a Web page presentation layout. Unfortunately, when a Web page changes its presentation layout, the API must be modified to prevent failure. Fundamental to having computer programs or agents implement reliable, large-scale interoperation of Web services is the need to make such services computer interpretable – to create a Semantic Web of services whose properties, capabilities, interfaces, and effects are encoded in an unambiguous, machine-understandable form.

To realize the vision of Semantic Web Services (McIlraith et al. 2001), creating semantic markup of Web services that makes them machine understandable and use-apparent is necessary. Equally important is the development of agent technology that exploits this semantic markup to support automated Web service composition and interoperability. Driving the development of the markup and agent technology is the automation of tasks that semantic markup of Web services will enable – most importantly, *service discovery, execution*, and *composition* and *interoperation*.

Semantic Web Services offer the possibility of highly flexible Web services architectures, where new services can be quickly discovered, orchestrated and composed into workflows.

Figure 14-10 highlights the Semantic Web Services (Fensel and Bussler 2002) that are intended to make the web realize its full potential.

The following discusses the various aspects of Semantic Web Services:

- *Semantic Web Services* define exhaustive description frameworks for describing Web Services and related aspects by means of Web Service Description Ontologies. They support ontologies as underlying data models to allow machine-based data interpretation and they define semantically driven technologies for the automation of the Web Service usage process.
- The following describes the *usage process* (Paolucci et al. 2003) for Semantic Web Services:
  1. *Publication*: This is to make available the description of the capability of a service. The capability of a service declares the various parameters associated with it and the functionality it has to offer to the outside world (of agents and other services).
  2. *Discovery*: This is locating different services suitable for a given task. It involves automatically locating Web services that provide a particular service and that adhere to requested properties (task). A user might say, for example, "Find a service that

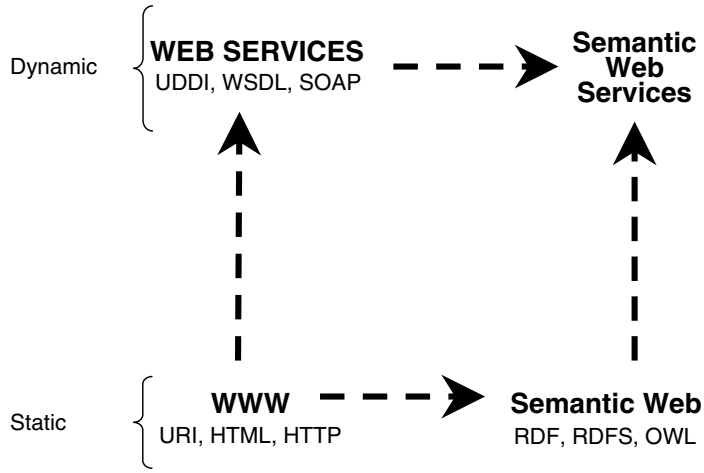


Figure 14-10 [1]Web, Semantic Web, Web Services and Semantic Web Services. (For technical details on Web Services please refer to Chapter 6.)

sells airline tickets between Vienna and Amsterdam and that accepts payment by Master card." Currently, a human must perform this task, first using a search engine to find a service and then either reading the Web site associated with that service or executing the service to see whether it adheres to the specified task. With semantic markup of services, we can specify the information necessary for Web service discovery as computer-interpretable semantic markup at the service Web sites, and a service registry or (ontology-enhanced) search engine can automatically locate appropriate services.

3. *Selection*: This task implies choosing the most appropriate service for a given task assuming that, in principle, a potentially large number of services can fulfill the requirements of the specified task. Here the criterion for selection of a particular service from the pool can be universal or user specific with parameters such as money involved, time constraints, etc.
4. *Composition*: This task implies combining services to achieve the specified goal. Sometimes a single service may not be sufficient to fulfill the specified goal. It may require combining various services in a specific order where the outcome of one or more service executions will be the input to another set of services. Currently, if some task requires a composition of web services that need to interoperate, then the user must select those web services, manually specify the composition, ensure that any software for interoperation is custom-created, and provide the input at choice points (for example, selecting a flight from several options). With semantic markup of web services, the information necessary to select, compose, and respond to services is encoded at the service web sites.

1 This figure is taken from the tutorial presentation on "Semantic Web Services" in the European Semantic Web Conference, Heraklion, Crete, 2005.

5. *Mediation*: This is a necessary functionality aimed at solving semantic mismatches, which can happen at the data level, protocol level and/or at the process level.
6. *Execution*: This is the invocation of services following programmatic conventions. This supports *Monitoring* (control over the execution process), *Compensation* (providing transactional support and mitigating unwanted effects), *Replacement* (facilitating the substitution of services by equivalent ones) and *Auditing* (verification that the execution took place as expected).

- *Semantic Web Service Ontologies* were discussed in detail in the previous sections. However for the case of ontologies for the semantic web services the following two stand out:
  - *OWL-S: Semantic Markup for Web Services 2004*: OWL-S is an OWL ontology to describe (semantic) web services. It does not aim at replacing existing web services standards but at providing a semantic layer over existing Web Services standards such as Web Services Description Language. It relies on WSDL for the invocation of web services and on Universal Description, Discovery and Integration for web services discovery. From the OWL-S perspective, a Semantic Web Service is provided by a *Resource*, presented by a *Service Profile*, described by a *Service Model* and supported by a *Service Grounding*. The Service Profile represents what the Service provides and capabilities of the Service; the Service Model describes how a service works and which are internal processes (Input; Preconditions; Outputs and Results) of the service. Processes can be of the types Atomic, Simple, or Composite. Finally, the Service Grounding builds upon WSDL descriptions to define the message structure and physical binding layer and maps an XML schema to OWL concepts.
  - *Web Service Modeling Ontology (WSMO)*: Feier and Domingue (2005) is a conceptual model for Semantic Web Services. It is an ontology for core elements for the services and consists of a formal description language (Web Services Modeling Language) and an execution environment (Haller et al. 2005). WSMO defines the modeling elements for Semantic Web services based on the conceptual grounding specified in the four main components of the Web Service Modeling Framework (Fensel and Bussler 2003):
    - **Ontologies** provide the formal semantics to the information used by all other components.
    - **Goals** specify objectives that a client might have when consulting a Web service.
    - **Web services** represent the functional (and behavioral) aspects, which must be semantically described in order to allow semi-automatic use.
    - **Mediators** used as connectors to provide interoperability facilities among the other elements.
- Web Services have an aspect of *capability*, which defines their functionality in terms of pre- and post-conditions, assumptions and effects. A Web Service defines one and only one capability. The *interface* of a Web Service provides further information on how the functionality of the Web Service is achieved. It contains:
  - A *Choreography*, which describes the communication pattern, that allows to one to consume the functionality of the Web service, named *choreography.* In other words *Choreography* is how to interact with a service to consume its functionality.
  - An *Orchestration*, which describes how the overall functionality of the Web service is achieved by means of cooperation of different, Web service providers, named

orchestration. In other words *Orchestration* is how service functionality is achieved by aggregating other web services.

The service paradigm will be a crucial precondition for the industrial use of the Semantic Web. The semantic description of Web Services will allow better advertising and subsequently, better discovery of Web Services and more importantly supply a better solution for the selection, invocation, composition, monitoring and interoperation of Web Services.

14.3.5 Integration into Web Engineering

If we want to expand the method for the development of Web applications by a semantic component, it is useful to remember how a "semantic" Web application differs from a "traditional" Web application:

- *Agents*: The first fundamental difference is the mandatory assumption that a semantically marked up information space will be visited by software agents, and that these agents can extract "knowledge" from this space with or without the help of an information provider. This means that the information space should have an agent-supporting architecture.
- *Knowledge representation*: The second fundamental assumption is that the information space provides a high degree of semantic comparability with other information spaces. This means that extremely high requirements are posed on standardization of the terminology and its interpretation options. This can be facilitated by using a general knowledge representation language that allows converting semantic annotations into predicate logic and eventually into decidable statements. Consequently, a Semantic Web application will use a standardized language for knowledge representation.
- *Knowledge modeling*: A Web application that supports semantic annotation has to be fitted with an ontology during its design phase. In the simplest case, we would take an existing ontology. In the most complicated case, we would use a consolidation process within the user group of the relevant application domain to work out an ontology for the first time ever.
- *Semantic annotation*: If we decide to use an existing ontology, we will have to structure and label the information space based on this ontology. For complex applications, this will most likely be done by experts, who should be given appropriate tools, i.e., annotation editors, for the application's maintenance environment. For simpler and highly structured applications, it may be possible to automate part of the annotation work.
- *Knowledge storage and query*: An information space is characterized by its heterogeneous data elements (images, graphics, text, audio and video files). While structural and navigational concepts are generally well understood, it is much harder to classify the role of meta-data and knowledge concepts as well as queries.

This expansion of a general architecture results in the following additional steps for the development method of Semantic Web applications:

- Adaptation of the system to software agents.
- Selection of a knowledge representation language with tool support, including adaptation of the tools to the concrete environment.

- Creation of an application-specific ontology (if required).
- Manual and/or automatic annotation of the information space that should be covered by the Web application under development.

Since the elements of Semantic Web applications are more closely related to the building of knowledge-based systems than to the building of hypertext systems, one has to expect a significant shift in required development expertise once Semantic Web applications become more widespread. On the other hand, if the Semantic Web proves a success, then there will be an increasing number of vendors who will offer innovative, easy-to-use development tools.

14.4 Tools

Environments designed to build agents and editors to support the creation of ontologies and tools for semantic markup of Web pages are of prime interest for the development of Semantic Web applications. However, we should be aware of the fact that most tools currently available are research prototypes, since there are no mature technologies yet.

Agent Frameworks

One of the most time-consuming development processes is the construction of agent systems. For FIPA-compatible agent systems alone, there are currently well over a dozen development tools, which are surely worth being tested before trying to design a new agent system from the scratch. JADE and FIPA-OS are good examples of such development environments.

Bellifemine et al. (2001) describe JADE, a development environment for multi-agent systems based on Java and on the FIPA standards (`http://sharon.cselt.it/projects/jade`). The system has been under development since 1999, and it includes tools to build agents and to manage and distribute agents.

Similar to JADE, FIPA-OS is also designed to methodologically support the construction and management of agent systems by providing appropriate tools (`http://fipa-os.sourceforge.net`). FIPA-OS is an open-source project. At the end of 2001, an agent framework called Micro-FIPA-OS was introduced for mobile applications on PDAs and similar devices. In the area of Semantic Web Services, a number of research frameworks are emerging. One of the most prominent toolsets is provided by the groups developing the Web Service Modelling Ontology (WSMO, `http://www.wsmo.org`).

Ontology Editors

If there is no ontology available for a specific application domain, or if we want to further develop an existing ontology, we will have to use a tool to maintain the hierarchical order of the terms and their definitions. This is where ontology editors come in. An ontology editor supports us in defining a conceptual hierarchy, including attributes and relations between the concepts, and it assists us in creating synonym lists and multi-lingual thesauruses. Most of the more recent tools support RDF Schema and DAML + OIL or OWL by way of import and export functions. Examples of such editors include Protege2000 (Noy et al. 2001), OntoEdit (Sure et al. 2002), and OilEd (Bechhofer et al. 2001).

Annotation Tools

Currently, the semantic markup of Web pages is still a bottleneck hindering the proliferation of the Semantic Web. It should basically be possible to automate the semantic markup. This is well conceivable for new Web applications, for instance, by using a content management system based on an ontology. In this case, the contents would be semantically marked up by indexing them according to that ontology. This can be done as early as during the contents preparation. Things get more difficult if very complex statements have to be represented, such as statements for operating instructions or scientific discussions. In these cases, we will have to fall back on manual annotation, and the annotator will have to stick to guidelines specifying how the underlying ontology should be used. Examples of such tools include the Annotea project (`http://www.w3.org/2001/Annotea/`). In its current version, Annotea provides mechanisms for general annotations based on a pre-defined RDF schema. In addition, there are electronic tools for scientific group work (Kirschner et al. 2003), and a general multimedia annotation tool specialized for culture and arts (`http://www.cultos.org`). There has also been a large body of work on annotating scholarly discourse (`http://kmi.open.ac.uk/projects/scholonto/scholonto-archive.html`).

14.5 Outlook

With the integration of knowledge representation, the Web hits the limits of its original design philosophy. As we all know, the original idea was to structurally map knowledge connections between documents with related contents or document sections to overcome physical boundaries (article A refers to an experiment described in B, and the data for B can be read in the log or minutes of C, and each of these documents resides in a different directory and possibly even on a different server). Consequently, the original philosophy was to transform existing knowledge into a "reachable" structure. Hypertext navigation was then primarily intended to substitute search processes. The new form of the Web, however, promises new qualities: by explicitly describing the relationship of articles, experiments, and experiment logs, we can check and recombine statements (in articles), arrangements (in experiments), and results (in protocols). This opens up new avenues, for instance in medical research, because stronger structuring of information facilitates new evaluation methods (e.g., clinical meta-studies).

The philosophy of the Semantic Web is based on converting the contents of nodes contained in this reachable structure so that a node can be interpreted by machines. This is the point where the conceptual difference to the traditional Web comes to light. In the traditional Web, the function of any hyperlink connection was of type "`GO TO`", e.g. `GO TO <experiment>` and `GO TO <results log>`. The typing of the Web pages (e.g. "experiment" or "results-log") was interpretable by humans only. In the Semantic Web, the article becomes a self-describing entity ("I'm a scientific article based on experiment B and the associated log C"). The consequences are fundamental: first, if the referenced entities B and C have unique names, it doesn't matter where the documents are actually stored as long as they are somewhere where they can be found (by machines). The effect is that the explicit linking of information becomes secondary, while the effectiveness of the calculable links (based on self-describing entities) becomes primary. Second, if self-description (gradually) improves, it will eventually be unnecessary, because it will be deducible from the content itself. Once arrived at this point, each new piece of content

will become part of a future knowledge universe, because machines link the content to related contents in calculable connections. When this will be possible is speculative in that it requires the understanding of languages, and additionally the ability (of machines!) to imagine (rather, calculate) possible worlds. AI research set itself these goals about fifty years ago, and has not achieved them to this day. At the same time, however, AI has made important contributions to IT as we know it today. Databases and search engines use many of the principles formulated by AI researchers during recent decades. So, a renaissance of the "great challenges" could give important impulses for future technologies, though the most ambitious goals will probably be hard to reach during the next fifty years, too. However, we also have to bear in mind that the enormous computer powers of current systems enable solutions to move into the feasible range. Though these solutions do not try to imitate human thinking, they can achieve very remarkable performances in other ways (often by brute-force evaluation).