The Office of Naval Research Workshop and Conference, September 8-9, 2004. ***RESU84-A***

Interoperability and the Need for Intelligent Software A Historical Perspective

Jens Pohl, Ph.D.

Executive Director, Collaborative Agent Design Research Center (CADRC)
California Polytechnic State University (Cal Poly)
San Luis Obispo, California, USA

With the objective of defining the ***interoperability*** theme of this year's conference it is the purpose of this paper[1] to trace the evolution of intelligent software from data-centric applications that essentially encapsulate their data environment to ontology-based applications with automated reasoning capabilities. The author draws a distinction between human intelligence and component capabilities within a more general definition of intelligence; - a kind of intelligence that can be embedded in computer software. The primary vehicle in the quest for intelligent software has been the gradual recognition of the central role played by data and information, rather than the logic and functionality of the application. The three milestones in this evolution have been: the separation of data management from the internal domain of the application; the development of standard data exchange protocols such as XML that allow machine interpretable structure and meaning to be added to data exchange packages; and, the ability to build information models that are rich in relationships and are thereby capable of supporting the automated reasoning capabilities of software agents.

The author suggests that the vision of a Semantic Web environment in which ontology-based Web services with intelligent capabilities are able to discover each other and individually or in self-configured groups perform useful tasks, is not only feasible but imminently realizable. The capabilities of an experimental proof-of-concept system featuring semantic Web services that was demonstrated during the 2002 meeting of this annual conference series is described in summary form.

The Concept of 'Intelligence'

Before we can proceed with the theme of this paper it is necessary to briefly discuss the concept of ***intelligence*** and the sense in which this concept is applied by the author. There are those that have advanced strong arguments that intelligence is the province of living creatures and that machines, such as electronic computers, do not and will never display any truly intelligent capabilities (Dreyfuss 1979 and 1997, Dreyfuss and Dreyfuss 1986, Lucas 1961, Searle 1980 and 1992). In most cases these arguments are based on the premise that intelligent behavior is closely associated with the human body and mind, and that the powerful notions of common sense and intuition are essential ingredients of intelligence. It is not the purpose of this paper to attempt to counter these arguments or even take sides in this debate.

Instead, the author wishes to advance another view of intelligence, namely that human intelligence and intelligence are not synonymous. We human beings are a decidedly self-centered species. We tend to view our capabilities and our interactions with our surroundings

[1] This paper is based on a keynote address delivered at the 16th International Conference on Systems Research, Informatics and Cybernetics (InterSymp-2004), held in Baden-Baden, Germany, July 29 – August 5, 2004.

from a very personal point of view. It is therefore not surprising that we should consider intelligence, which is essentially our most powerful asset, to be restricted to living creatures among whom we believe ourselves to reign supreme.

Webster's Dictionary (Random 1999) defines intelligence as the "… capacity for learning, reasoning, and understanding;". This definition suggests that there are component capabilities that contribute to the concept of intelligence. Further, these component capabilities are not necessarily equally powerful. In other words, it may be argued that there are levels of intelligence and that at the lowest level such capabilities must include at least the ability to remember. Higher levels of intelligence include reasoning, learning, discovering, and creating. Certainly at least some of these intelligent capabilities can be embedded in computer software. For example, computers excel at storing and recalling data in virtually unlimited quantities and over very long periods of time. Computers can reason about data quite effectively, if adequate context is made available with the data. Also, computers have been shown to have learning-like capabilities, and computers can discover information through associations and pattern matching.

There is no intention by the author to suggest that computer intelligence is equal or even similar to human intelligence, but rather that computer intelligence and human intelligence may be applied in parallel to complement each other. Furthermore, a strong case can be made in support of the view that there is an urgent need for intelligent computer capabilities due to the mounting expectations of accuracy, quality and timeliness in a globally connected environment of rapidly increasing complexity.

The Need for Software Intelligence

There are essentially two compelling reasons why computer software must increasingly incorporate more and more 'intelligent' capabilities. The first reason relates to the current data-processing bottleneck. Advancements in computer technology over the past several decades have made it possible to store vast amounts of data in electronic form. Based on past manual information handling practices and implicit acceptance of the principle that the interpretation of data into information and knowledge is the responsibility of the human operators of the computer-based data storage devices, emphasis was placed on storage efficiency rather than processing effectiveness. Typically, data file and database management methodologies focused on the storage, retrieval and manipulation of data transactions, rather than the ***context*** within which the collected data would later become useful in planning, monitoring, assessment, and decision-making tasks.

The second reason is somewhat different in nature. It relates to the complexity of networked computer and communication systems, and the increased reliance of organizations on the reliability of such information technology environments as the key enabler of their effectiveness, profitability and continued existence.

The Data-Processing Bottleneck

This requires further explanation, as a fundamental issue and one of the primary forces driving the evolution of software intelligence. The design of any information system architecture must be based on the obvious truth that the only meaningful reason for capturing and storing data is to utilize them in some planning or decision-making process. However for data to be useful for planners and decision makers they have to be understood in context. In other words, data are just numbers and words that become meaningful only when they are viewed within a situational framework. This framework is typically defined by associations that relate data items to each other and peripheral factors, which influence the meaning of the data in a particular situation. Succinctly stated, numbers and words (i.e., data) found within a rich set of relationships become information, which provides the necessary context for interpreting the meaning of the data, the recognition of patterns, and the formulation of rules, commonly referred to as knowledge.

The larger an organization the more data it generates itself and captures from external sources. With the availability of powerful computer hardware and database management systems the ability of organizations to store and order these data in some purposeful manner has dramatically increased. However, at the same time, the expectations and need to utilize the stored data in monitoring, planning and time-critical decision-making tasks has become a major human resource intensive preoccupation. In many respects this data-centric focus has become a bottleneck that inhibits the ability of the organization to efficiently and effectively accomplish its mission.

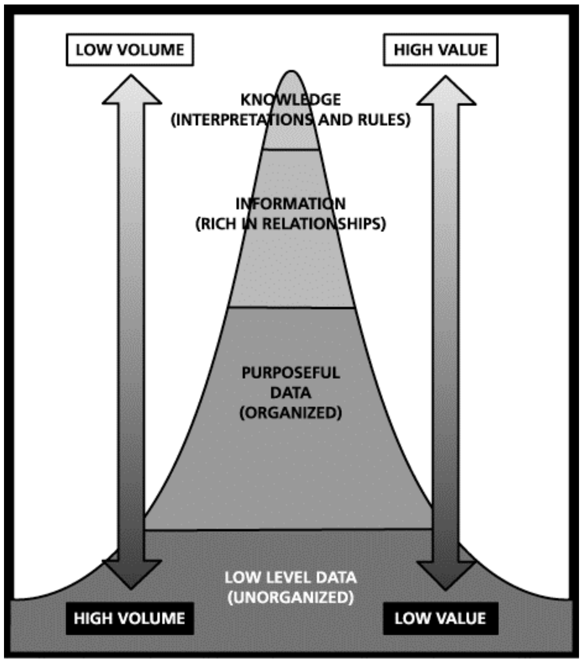

LOW VOLUME
HIGH VALUE
KNOWLEDGE (INTERPRETATIONS AND RULES)
INFORMATION (RICH IN RELATIONSHIPS)
PURPOSEFUL DATA (ORGANIZED)
LOW LEVEL DATA (UNORGANIZED)
HIGH VOLUME
LOW VALUE

Figure 1: Transition from data to knowledge

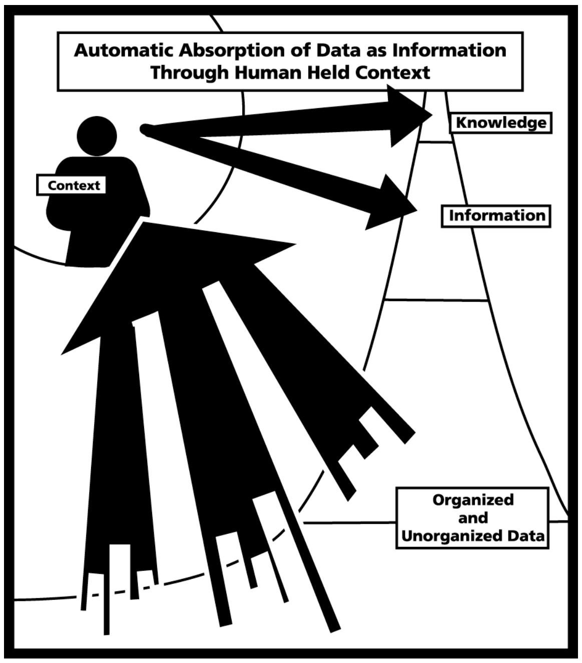

Automatic Absorption of Data as Information Through Human Held Context
Knowledge
Context
Information
Organized and Unorganized Data

Figure 2: Human interpretation of data

The reasons for this bottleneck are twofold. First, large organizations are forced to focus their attention and efforts on the almost overwhelming tasks involved in converting unordered data into purposefully ordered data (Figure 1). This involves, in particular, the establishment of gateways to a large number of heterogeneous data sources, the validation and integration of these sources, the standardization of nomenclatures, and the collection of data elements into logical data models. Second, with the almost exclusive emphasis on the slicing and dicing of data, rather than the capture and preservation of relationships, the interpretation of the massive and continuously increasing volume of data is left to the users of the data (Figure 2). The experience and knowledge stored in the human cognitive system serves as the necessary context for the interpretation and utilization of the ordered data in monitoring, planning and decision-making processes. However, the burden imposed on the human user of having to interpret large amounts of data at the lowest levels of context has resulted in a wasteful and often ineffective application of valuable and scarce human resources. In particular, it often leads to late or non-recognition of patterns, overlooked consequences, missed opportunities, incomplete and inaccurate assessments, inability to respond in a timely manner, marginal decisions, and unnecessary human burn-out. These are symptoms of an incomplete information management environment. An environment that relies entirely on the capture of data and the ability of its human users to add the relationships to convert the data into information and thereby provide the context that is required for all effective planning and decision-making endeavors.

A more complete information management environment considers data to be the bottom layer of a three-layer architecture, namely:

> A ***Data Layer*** that integrates heterogeneous data sources into accessible and purposefully ordered data. It typically includes a wide variety of repositories ranging from simple textual files to databases, Data Portals, Data Warehouses, and Data Marts.
>
> A ***Mediation Layer*** that defines the structure of the data sources (i.e., logical data models), data transfer formats, and data transformation rules. The two principal purposes of the Mediation Layer are to facilitate the automated discovery of data and to support the mapping of data to information. In other words, the Mediation Layer serves as a registry for all definitions, schemas, protocols, conventions, and rules that are required to recognize data within the appropriate context. The Mediation Layer also serves as a translation facility for bridging between data with structural relationships (e.g., based on a logical data model) and information that is rich in contextual relationships.
>
> An ***Information Layer*** that consists of many functionally oriented planning and decision-assistance software applications. Typically, these applications are based on internal information models (i.e., object models or ontologies) that are virtual representations of particular portions of the real world context. By providing context, the internal information model of each application is able to support the automated reasoning capabilities of rule-based software agents.

In such a three-layered information management environment the Mediation Layer continuously populates the information models of the applications in the Information Layer with the data changes that are fed to it by the Data Layer. This in turn automatically triggers the reasoning capabilities of the software agents. The collaboration of these agents with each other and the human users contributes a powerful, near real-time, adaptive decision-support environment. The agents can be looked upon as intelligent, dynamic tools that continuously monitor changes in the real world. They utilize their reasoning and computational capabilities to generate and evaluate courses of action in response to both real world events and user interactions. As a result the human user is relieved of many of the lower level filtering, analysis, and reasoning tasks that are a necessary part of any useful planning and problem solving process. However, just as importantly, the software agents continuously and tirelessly monitor the real world execution environment for changes and events that may impact current or projected plans.

The Increasing Complexity of Information Systems

The economic impact on an organization that is required to manually coordinate and maintain hundreds of interfaces between data-processing systems and applications that have no 'understanding' of the data that they are required to exchange, is enormous. Ensuing costs are not only related to the requirement for human resources and technical maintenance (normally contracted services), but also to the indirect consequences of an information systems environment that has hundreds of potential failure points.

Recent studies conducted by IBM Corporation and others have highlighted the need for autonomic computing as the organizational expectations and dependence on information services leads to more and more complex networked computer solutions (Ganek and Corbi 2003). In the commercial sector "…it is now estimated that at least one-third of an organization's IT (Information Technology) budget is spent on preventing or recovering from crashes" (Patterson et al. 2002). Simply stated (Figure 3), autonomic computing utilizes the 'understanding' that can be represented within an information-centric software environment to allow systems to automatically: (1) reconfigure themselves under dynamically changing conditions; (2) discover, diagnose, and react to disruptions; (3) maximize resource utilization to meet end-user needs and system loads; and, (4) anticipate, detect, identify, and protect themselves from external and internal attacks.

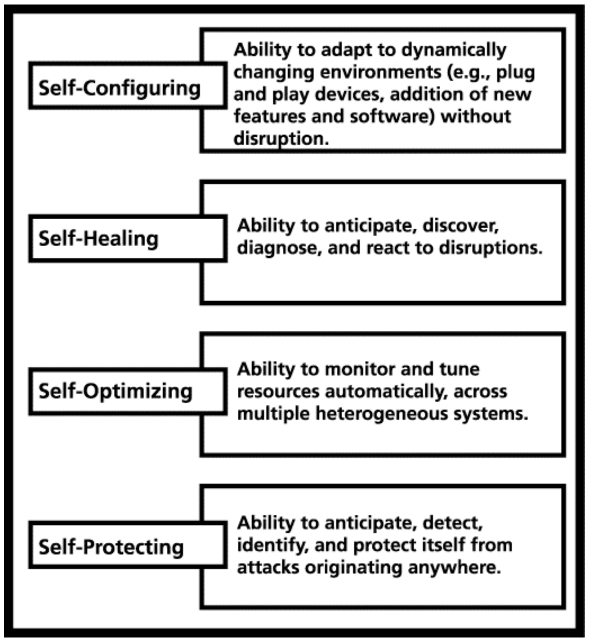

Self-Configuring
Ability to adapt to dynamically changing environments (e.g., plug and play devices, addition of new features and software) without disruption.
Self-Healing
Ability to anticipate, discover, diagnose, and react to disruptions.
Self-Optimizing
Ability to monitor and tune resources automatically, across multiple heterogeneous systems.
Self-Protecting
Ability to anticipate, detect, identify, and protect itself from attacks originating anywhere.

Figure 3: Desirable autonomic capabilities

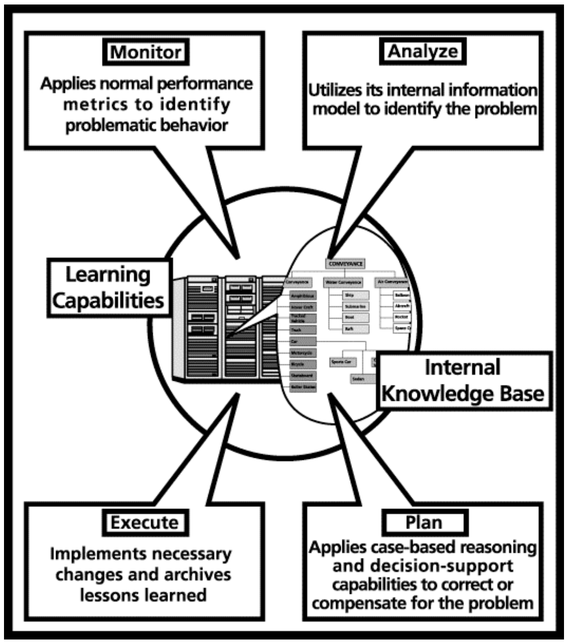

Monitor
Applies normal performance metrics to identify problematic behavior
Analyze
Utilizes its internal information model to identify the problem
Learning Capabilities
Internal Knowledge Base
Execute
Implements necessary changes and archives lessons learned
Plan
Applies case-based reasoning and decision-support capabilities to correct or compensate for the problem

Figure 4: Autonomic self-healing requirements

These same studies have found that more than 40% of computer system disruptions and failures are due to human error. However, the root cause of these human errors was not found to be lack of training, but system complexity. When we consider that computer 'downtime' due to security breaches and recovery actions can cost as much as (US)$2 million per hour for banks and brokerage firms, the need for computer-based systems that are capable of controlling themselves (i.e., have autonomic capabilities) assumes critical importance.

A core requirement of autonomic computing is the ability of a computer-based information system to recover from conditions that already have caused or will likely cause some part(s) of the system to fail. As shown in Figure 4, this kind of self-healing capability requires a system to continuously monitor itself so that it can identify, analyze and take mitigating actions, preferably before the disruption takes place. In addition, the system should be able to learn from its own experience by maintaining a knowledge base of past conditions that have caused malfunctions and the corrective measures that were taken.

In summary, the continued expansion of networks (e.g., the Internet and its successors) will provide seamless connectivity among countless nodes on a global scale. While the collection of data has already increased enormously over the past decade, the availability of such a global network is likely to increase the volume of data by several orders of magnitude. Such a volume of raw data is likely to choke the global network regardless of any advances in communication and computer hardware technology. To overcome this very real problem there is a need to collect data in context so that only the data that are relevant and useful are collected and transmitted within the networked environment. Most (if not all) of the necessary filtering must be achieved automatically for at least three reasons. First, organizations cannot afford to utilize human resources for repetitive tasks that are tedious and require few human intellectual skills. Second, even if an organization could afford to waste its human resources in this manner it would soon exhaust its resources under an ever-increasing data load. Third, it does not make sense for an organization to 'burn-out' its skilled human resources on low-level tasks and then not have them available for the higher-level exploitation of the information and knowledge generated by the lower level tasks.

Finally, the increased reliance on computer-based information systems mandates a level of reliability and security that cannot be achieved through manual means alone. The alternative, an autonomic computing capability, requires the software that controls the operation of the system to have some understanding of system components and their interaction. In other words, autonomic computing software demands a similar internal information-centric representation of context that is required in support of the knowledge management activities in an organization. In both cases the availability of data in context is a prerequisite for the reasoning capabilities of software agents (i.e., the automatic interpretation of information by the computer).

A Framework for Assessing Software Capabilities

Just like the initial conception and implementation of computing devices was driven by the human desire to overcome the limitations of manual calculation methods, the advancements in computing technology during the past 50 years have been driven by the desire to extend the usefulness of computer-based systems into virtually every human activity. It is not surprising that after several orders of magnitude increases in hardware performance (i.e., computational speed and data storage capacity (Pohl 1998)) had been achieved, attention would gradually shift from hardware to software.

Increasingly software is being recognized as the vehicle for computers to take over tasks that cannot be completely predefined at the time the software is developed. The impetus for this desire to elevate computers beyond data-processing, visualization and predefined problem-solving capabilities, is the need for organizations and individuals to be able to respond more quickly to changes in their environment. Computer software that has no 'understanding' of the data that it is processing must be designed to execute predefined actions in a predetermined manner. Such software performs very well in all cases where it is applied under its specified design conditions and performs increasingly poorly, if at all, depending on how much the real world conditions vary from those design specifications. Instead, what is needed is software that incorporates tools, which can autonomously adapt to changes in the application environment.

Adaptable software presupposes the ability to perform some degree of automated reasoning. However, the critical prerequisite for reasoning is the situational context within which the reasoning activity is framed. It is therefore not surprising that the evolution of computer software in recent years has been largely preoccupied with the relationship between the computational capabilities and the representation of the data that feed these capabilities. One could argue that the historical path from unconnected atomic data elements, to data structures, relational databases, data objects, object-oriented databases, object models, and ontologies, has been driven by the desire to provide information context in support of automated reasoning capabilities.

However, to be able to present a true historical perspective of the evolution of software it is necessary to take into account a more comprehensive set of criteria. In fact, there are several factors that have in the past and are continuing to contribute to the evolution of intelligent software. This section will attempt to establish a set of categorization criteria to serve as a framework for tracing the capabilities of software. Since these capabilities are closely related to the design and implementation of the computer-based environment within which the software is required to operate, the proposed framework will utilize ***system architecture*** as a yardstick and milestone component. The following eight system architectures have been selected to serve as milestones for the assessment of software capabilities:

- ***Single data-centric applications*** that operate in a stand-alone mode and receive data from user interaction and other closely coupled sources (e.g., data files and dedicated databases).
- ***Confederation of linked data-centric applications*** with application-to-application data bridges. Also described as 'stove-pipe' systems because the system components are essentially hardwired to only work together within their confederation.
- ***Shared database systems*** consisting of multiple data-centric applications that are able to share data between themselves and a common repository, through application-to-database bridges. The repository may be either a single database or a distributed database facility.
- ***Distributed expert systems*** with dedicated knowledge bases (i.e., rules) and a single shared fact list (i.e., data).
- ***Distributed static information-based applications*** with collaborative agents, capable of exchanging data with external data-centric applications.

- ***Distributed static information-sharing applications*** with collaborative agents, capable of interoperating at the 'information' level with other ontology-based applications and capable of exchanging data with external data-centric applications.
- ***Distributed extensible information-sharing applications*** with collaborative agents, capable of interoperating at the 'information' level with other ontology-based applications and capable of extending their internal information representation (i.e., ontology) during execution.
- ***Semantic Web services*** capable of discovering other Web services and dynamically configuring themselves into distributed systems on an as-needed basis.

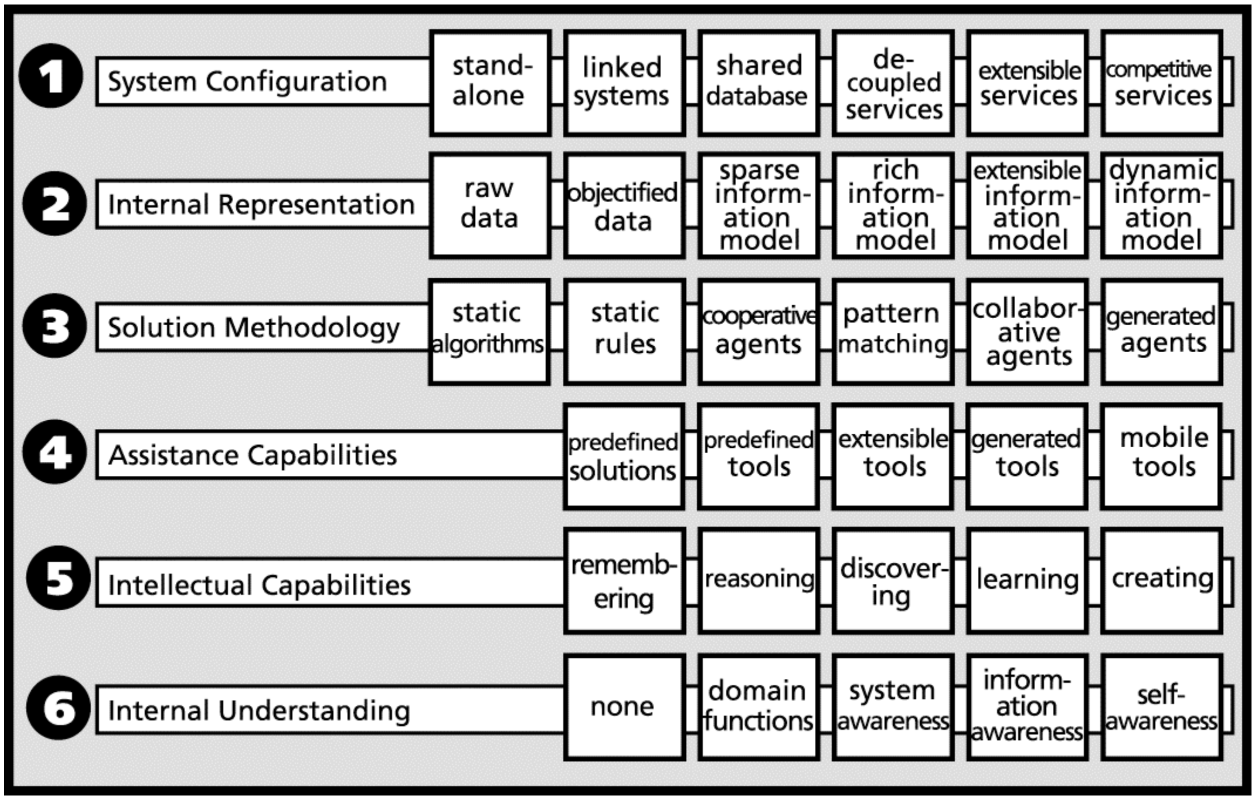


Figure 5: Software characterization categories and their capability criteria

The software capabilities that have been in the past or are still today prevalently applied in each of these system architectures are characterized within six capability groups as shown in Figure 5. While the first of these groups (i.e., Group (1) *System Configuration*) is intended to describe principal architectural features, the other five groups are focused on the degree to which the software is capable of representing and processing data with or without context in partnership with the human user. Fundamental in this respect is Group (2) *Internal Representation*. The manner in which an application represents the data that it is intended to manipulate essentially determines the level of software intelligence that the application is capable of supporting. Group (2) differentiates among applications that represent data without context (i.e., 'raw data' and 'objectified data'), applications that provide context in the form of a static information model (i.e., sparse information model' and 'rich information model') and applications with information models that are extensible during execution (i.e., 'extensible information model' and 'dynamic information model'). The remaining four groups address the general solution methodology available to the application, its decision-support capabilities, and the level of internal 'understanding' of its capabilities, activities and intrinsic nature. The divisions within each of the groups will be defined in more detail during the discussion of each of the eight system architectures.

The first system architecture for discussion (Figure 6) is representative of the typical early computer applications, namely a stand-alone application that receives all of its data from the user and/or data sources that are considered to be part of the application. Whether or not the data are treated as discrete elements or objects, the *Internal Representation* includes only a very limited set of relationships and therefore lacks context. Under these circumstances the *Assistance Capabilities* are limited to predefined solutions utilizing static algorithms, no internal understanding can be provided by the representation of data without relationships, and the *Intellectual Capabilities* of the software are restricted to 'remembering' since the data are stored in the computer. The second system architecture (Figure 7) adds data bridges between several data-centric applications. Each bridge is simply an application-to-application mapping of the data format of one application to the other. Therefore, the only capability that this architecture adds to the previously discussed architecture is that the *System Configuration* supports a confederation of tightly linked applications.

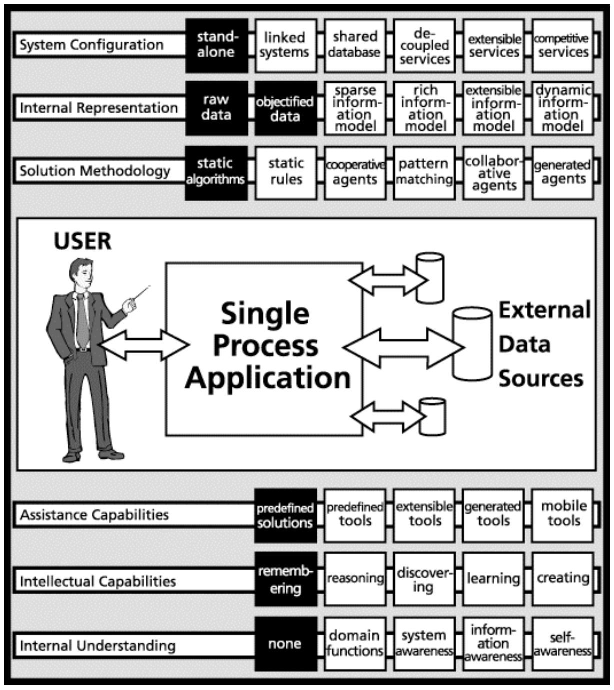


Figure 6: Single data-centric applications

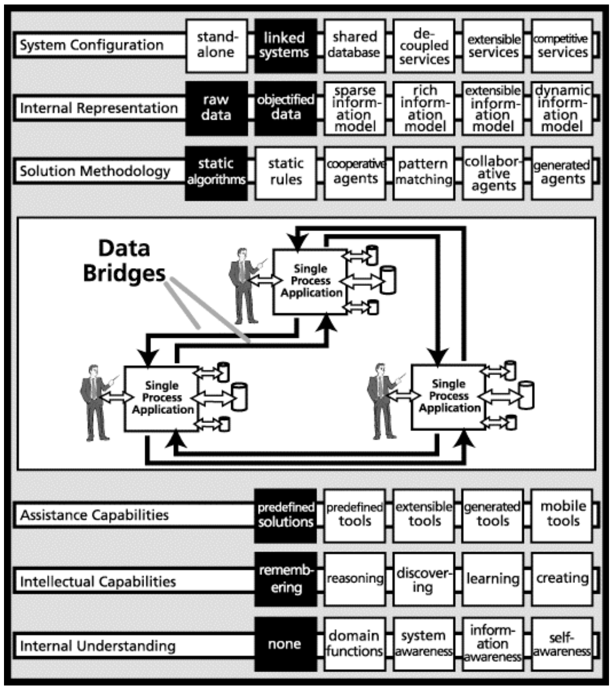


Figure 7: Confederation of linked data-centric applications

The shared database architecture (Figure 8) constitutes a major improvement over the first two system architectures by separating the data from the application and placing the former into a common repository that is external to all of the applications. The recognition that data and not the application should be the dominant component of a data-processing environment sets the stage for interoperability and intelligent software. However, it does not directly contribute any additional capabilities to the software criteria. The reason is the absence of data context, and this applies equally to the three system architectures discussed so far.

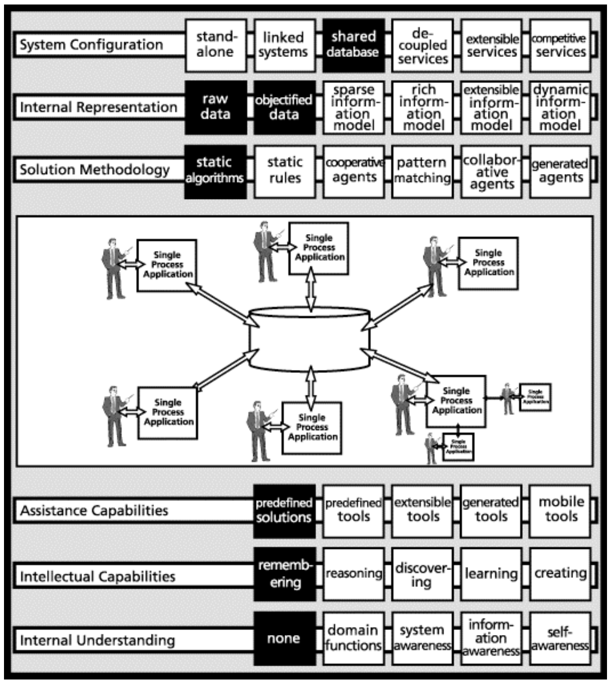


Figure 8: Shared database systems

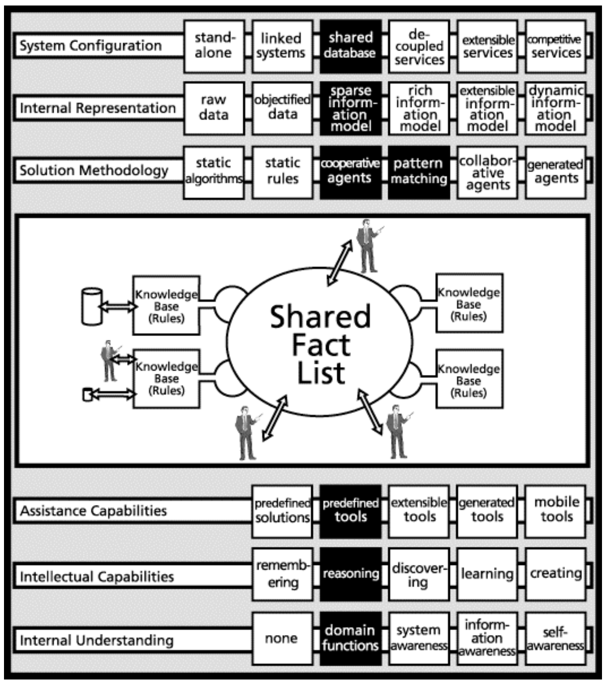


Figure 9: Distributed expert systems

The distributed expert system architecture shown in Figure 9 on the other hand, by virtue of its internal knowledge base of rules, driven by a shared repository of facts, adds several new capabilities to the software. Each knowledge base provides relationships and therefore represents a local component of what might be characterized as a sparse information model. This model provides adequate support for some form of automated reasoning within the typically narrow domain of each expert system. Although the expert systems (or agents) now operate as tools rather than predetermined solutions, their rules are nevertheless predefined and typically not extensible during execution.

For at least two reasons the concept of expert systems represents a milestone in the transition from data-processing to information-centric software. First, it showed that automated rule-based reasoning is in fact feasible and thereby allowed the field of artificial intelligence to regain some confidence after its earlier failures. Second, the largely opportunistic pattern-matching nature of an expert system laid the foundations for the notion of demon-like modules with particular data interests that could be triggered into action by data changes. Over the next decade these modules developed into *flexible* software agents that are *situated* in some environment and capable of *autonomous* actions (Wooldridge and Jennings 1995, Pohl et al. 2001 (32-33)). It was highly desirable for these agents to be capable of acting without the direct intervention of human users (or other agents), thereby providing the system with some degree of control over its own actions and internal state. The ability to achieve this level of autonomous behavior was greatly facilitated by situating the agent in a sufficiently well represented environment, which it can monitor and act upon. Triggered by its environment the agent is then able to respond to changes in the environment, exercise intiative through goal-directed reasoning capabilities, and utilize the services of other agents (including the human user) to supplement its own problem-solving capabilities in a collaborative fashion.

The desire for software agents to perform increasingly more valuable and human-like reasoning tasks focused a great deal of attention on the virtual representation of the real world environment in which the agent is situated. It became clear that the reasoning capabilities of a rule-based software agent depend largely on the richness of the virtual representation of this physical and conceptual environment. Taking advantage of the capabilities of object-oriented languages, which allow objects to be represented as classes with attributes and relationships, a new generation of application software with internal object-based information models was born (Figures 10, 11 and 12). These are often referred to as ontology-based applications and are typically distributed in nature.

It should be noted that the term ontology is commonly used rather loosely as a synonym for object model. Strictly speaking, however, the term ontology has a much broader definition. It actually refers to the entire knowledge in a particular field. In this sense, an ontology includes both an object model and the software agents that are capable of reasoning about information within the context provided by the object model (i.e., since the agents utilize business rules, which constitute some of the knowledge within a particular domain). In this paper the common use of the term ontology as an object model (i.e., context) is implied.

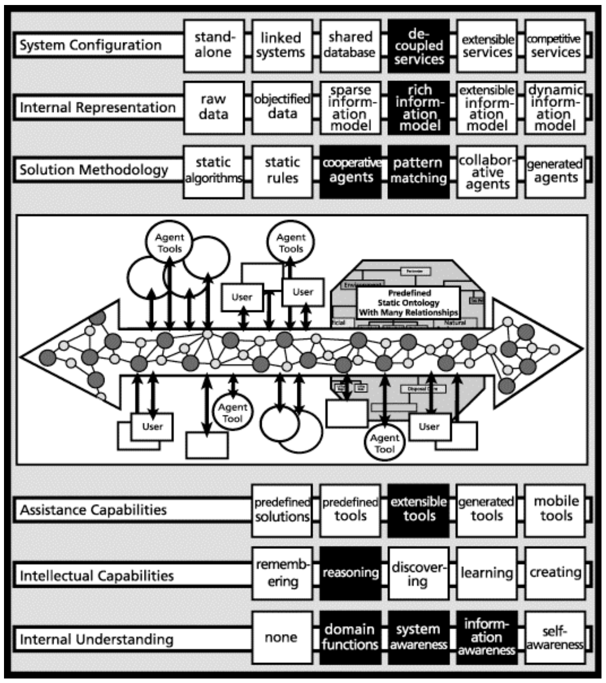


Figure 10: Information-based applications

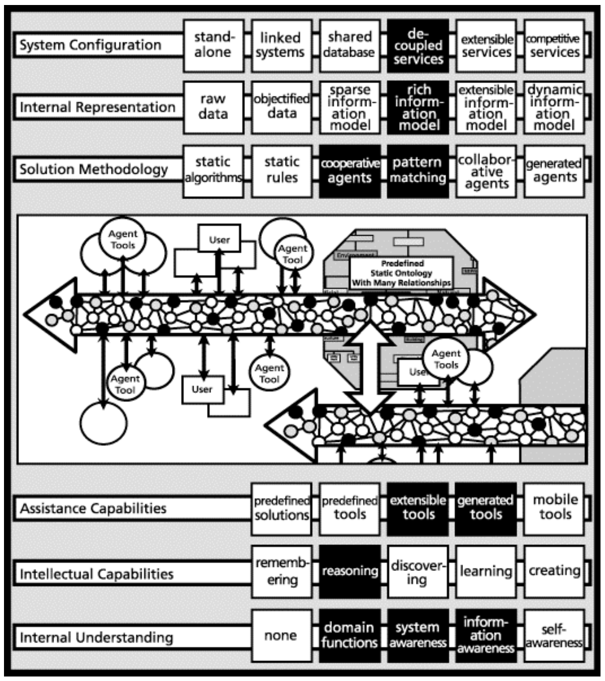


Figure 11: Information-sharing applications

The information-based architecture shown in Figure 10 typically consists of components (e.g., agents and user-interfaces) that communicate with each other through an information-serving collaboration facility. Each component includes a relevant portion of the ontology and a subscription profile of the kind of information that it is interested in receiving from this facility. Since the components have at least a limited understanding of the real world situation only the changes in the situation need to be communicated to them. While the existence of a subscription service obviates the need for computationally expensive queries in most cases, the ability to restrict the communication to changes in information also greatly reduces the amount of data that has to be exchanged. This applies equally to the information-sharing architecture and the extensible information architecture shown in Figures 11 and 12, respectively. Also, in all three of these software architectures system capabilities support (and promote) decoupled applications that interact via these services, which are accessed internally through clearly defined interfaces. Apart from simplifying the design and development of such applications, this allows services to be seamlessly replaced as long as the replacement service adheres to the same interface definition.

The principal differences among these three architectures are related to the adaptability and accessibility of the ontology within each of the information-centric systems. First, in both the information-based (Figure 10) and the information-sharing (Figure 11) architectures the ontologies are predefined at the time the applications are compiled and cannot be changed during execution. While it is certainly possible to build into an ontology some degree of flexibility that allows for the definition of variations of existing object types during execution, the context-based definition of new objects requires the application to be recompiled. In other words, the ontology is essentially static after the application has been compiled. In the extensible information-sharing architecture shown in Figure 12, an application is able to gain and share knowledge in its interactions with other applications that have similar capabilities, or with human users. The ability of an application to extend its understanding (i.e., to increase the context within which its agents are able to reason about changes in the real world situation) is still largely a subject of research. It involves the construction of context from data with sparse relationships, which intuitively would appear to be a poor approach. However, utilizing lexical (Fellbaum 1998) and algorithmic approaches developed in the natural language research domain (Pedersen and Bruce 1998), some surprisingly promising progress has been made in this area in the commercial arena (Cass 2004).

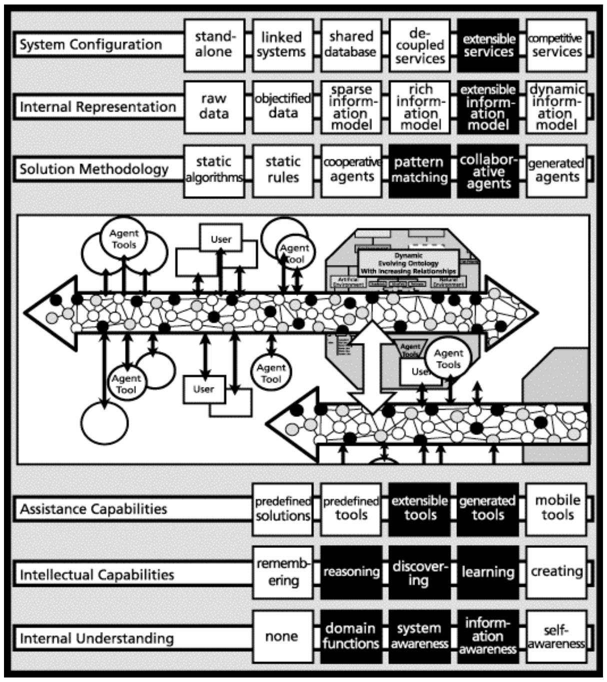


Figure 12: Extensible information-sharing applications

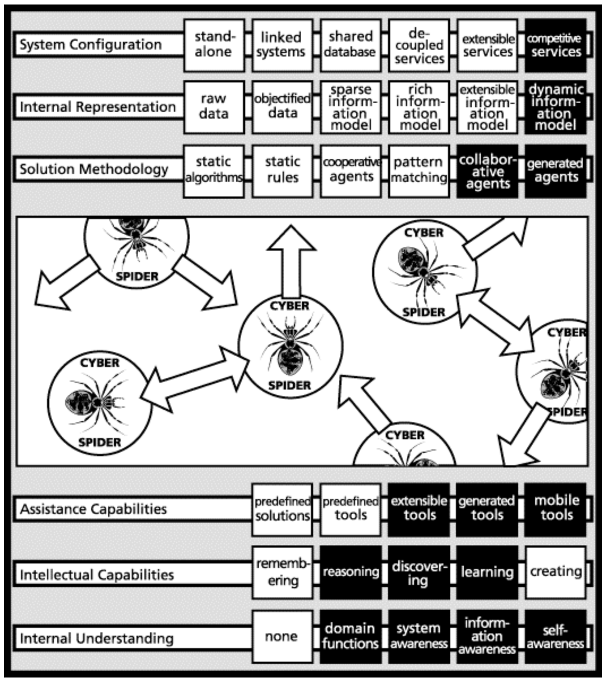


Figure 13: Semantic Web services

Second, in terms of accessibility, the subscription capabilities embedded in the components of an information-based system can be equally applied across multiple systems by having the information-serving collaboration facility of one system subscribe to the information-serving collaboration facility of another system. This is potentially a very powerful approach that allows information-centric systems to scale as clusters of networks within a networked environment.

The software architectures described so far (i.e., Figures 6 to 12) progressively evolved from stand-alone systems that encapsulate their own data, to systems that are able to share data based on predefined formats for data representation, to systems that incorporate rich but static information models and are able to support automated reasoning capabilities, to systems that are able to extend their internal information models in collaboration with similar ontology-based external systems. Within this evolutionary path the transition from data-based to information-based internal representation schemas is the enabling step that has endowed software with increasingly intelligent capabilities. However, the fundamental mechanism for achieving these capabilities is the ability to automatically reason about changes in the current state of the situation described by the information model. Once expert systems (Figure 9) had demonstrated that reasoning capabilities could be provided by conditional rules (i.e., a knowledge base of productions) and triggered by changes in a simple fact-list, it became clear that much could be gained by expanding the representational capabilities of the fact-list and incorporating in it many of the relationships that were formerly encoded in the rules of the knowledge base. This contributed to the formal separation within an application of the representation (i.e., object model or ontology) and the logic that is applied to this representation by agents. While initially most of the complexity of these ontology-based applications continued to reside in the agents, the availability of more powerful modeling concepts and tools is gradually allowing more and more of the complexity to be moved from the agents into the ontology. This suggests a trend that appears to mirror the earlier separation of an application from the data it is designed to manipulate (Figure 8), namely the separation of the information representation from the applications that incorporate reasoning capabilities. The combination of this trend with an information-centric Internet-like environment will cast applications into the role of capability-based services.

This is the emerging concept portrayed by the semantic Web services architecture shown in Figure 13. However, before describing this software architecture it is necessary to briefly discuss the architecture and capabilities of the existing data-centric Web services. They typically comprise a Web-Server that utilizes the Hyper-Text Transfer Protocol (HTTP) for communication, the Universal Description Discovery and Integration (UDDI) protocol as part of the standard definition of Web services registries, and a Registry that already contains an entry for the accessing application as well as any number of other Web services. UDDI is an international standard that defines a set of methods for accessing a Registry that provides certain information to an accessing application. For perhaps historical reasons UDDI is structured to provide information about organizations, such as: who (about the particular organization); what (what services are available); and, where (where are these services available).

The Simple Object Access Protocol (SOAP) defines a protocol for the direct exchange of data objects between software systems in a networked environment. It provides a means of representing objects at execution time, regardless of the underlying computer language. SOAP defines methods for representing the attributes and associations of an object in the Extensible

Markup Language (XML). It is actually a meta-protocol based on XML that can be used to define new protocols within a clearly defined, but flexible framework.

Web-Services are designed to be accessed by software. In the currently prevalent data-centric software environment they are generally clients to the middleware of data sources. The middleware collects the required data and sends them back to the Web service, which reformats the data using the SOAP protocol and passes them onto the requester. Depending on its original specifications, the requesting application will have the data downloaded on disk or receive them directly on-line. If the Web service is a data-centric application then a data-to-data translation must be performed in much the same way as is necessary when passing data between two data-centric applications.

Returning to the software architecture shown in Figure 13, the emphasis is on the word ***semantic***. In this architecture the semantics are embedded in an ontology, which provides the necessary context for automated reasoning. A semantic Web service, therefore, is an ontology-based application (may be mobile) with certain capabilities. Given a particular intent it seeks the services that it determines to be necessary for satisfying this intent. Having found one or more such Web services it self-configures itself with these discovered services into a temporary system. Depending on needs and circumstances this transitory system may reconfigure itself by discarding existing members when their capabilities are no longer needed, adding new members when other requirements arise, or dissolving itself altogether once it determines that its intent has been adequately executed.

To meet these capability objectives a semantic Web service reaches the highest-level criteria in all but one of the six software characterization categories shown in Figures 5 and 13. First, it operates in a competitive environment where it can select a service from several offering candidates, and presumably negotiate the terms of acceptance. Second, it incorporates a rich and extensible information model that will change dynamically as the semantic Web service discovers, collaborates with, and shares ontology fragments with its transitory partners. This provides the ability to create and maintain a desirable degree of common understanding within the self-configured system. Third, by virtue of this common understanding the agents of each member of the system are able to collaborate beyond the boundaries of the particular semantic Web service that they are housed in. Furthermore, any new agents that may be generated in response to a recently emerged need will likewise be able to collaborate globally within the system.

Fourth, the agents, which constitute the primary assistance capabilities of the system, become highly adaptable tools. They are extensible, they may be generated dynamically during execution to satisfy emerging new needs, and they can be implemented to operate in a mobile mode. Fifth, the collective intellectual capabilities of the system include the ability to discover capabilities that may be made available by external services and the ability to increase its understanding of context by extending the ontologies of one or more of its members through their interaction and the addition of new members to the system. It can be argued that this dynamic acquisition of new knowledge is a form of learning, however, it does not necessarily imply an ability to create new knowledge. Whether or not the semantic Web architecture will be able to create new knowledge is very much a matter of conjecture at this time.

Finally, in the *Internal Understanding* category the semantic Web architecture is rated to have the potential for reaching the highest criterion, 'self-awareness'. As further explanation it should be noted that this characterization category has been based entirely on the representational capabilities of ontologies, since the author is not aware of any alternative method for creating internal understanding in software. Ontologies are capable of not only representing physical objects such as buildings, conveyances (e.g., cars, boats, aircraft), supplies, weapons, and organizations, but also conceptual objects such as the notions of mobility, threat, privacy, security, consumability, and so on. This has been the predominant focus of ontologies to date. However, in addition, ontologies are able to represent the behavioral characteristics and relationships of the components of the software system itself. This is the domain of autonomic computing discussed previously, whereby a system is charged with continuously monitoring its own performance, exposure to intrusion, vulnerability to failure or degradation, and implementing remedies spontaneously as needs arise.

A third and much higher level of representation is the ability of a system to express to another system its nature, interests and capabilities. What is implied here is not simply an indication that this is a software system written in the Java computer language, supporting the following interface protocols, and listing explicitly defined capabilities. This kind of explicit introduction is similar to the directed search capabilities that are offered by the query facilities of any database management system available today. To fully support the requirements of 'discovery' the system should be able to communicate its nature, interests and capabilities in a conceptual manner. The analogy in the database domain is a conceptual search capability, where the target of the search is only vaguely defined as being something like something else and is expected to extend beyond the boundaries of any particular database or database management system (Pohl et al. 1999, 69-74). The ability to represent this kind of 'self-awareness' in an ontology appears to be well beyond current knowledge modeling capabilities.

The Semantic Web Initiative

It is unlikely that anyone predicted in the early 1970s when the Internet first appeared on the foundations of the ARPANET project funded by the U.S. Department of Defense Advanced Research Projects Agency (DARPA) that some 30 years later in 2003 the Internet would be used on a regular basis by more than 600 million people and serve as the preferred medium for close to (US)$4 trillion in business transactions. However, although the Internet provides almost instant global connectivity and potential access to an enormous volume of information, all of that information is stored in a low-level form as data. As a result, even the most powerful search engines can do little more than pattern-match on keywords as they attempt to retrieve user requested information. The product of such data searches is typically hundreds of information source references that may or may not be useful to the human user. The latter may then have to spend hours reviewing each source to determine whether it is relevant to the purpose of the search. This was not the intention of the creators of the World Wide Web (Berners-Lee and Fischetti 1999).

There is a valid concern that the more successful the Internet becomes in providing global connectivity to millions of users, with a corresponding exponential growth in the availability of information, the less useful it will become as a source of information. Succinctly stated the evolution of the Internet, like software systems in general, has been driven by the ability of computers to rapidly manipulate vast amounts of data without any understanding of the meaning of the data being processed. The vision of the Semantic Web is intended to overcome this serious deficiency by making the information on the World Wide Web understandable by computer software. Signs of this vision have become evident with the increasing interest in adding semantics to data.

The historical development of data manipulation and storage techniques first showed a preoccupation with efficiency, leading to the deletion of context in favor of the arrangement of data into neatly packaged records. This appeared to be a perfectly logical approach in line with the notion that the application, and not the data, is the enabler of the desired functionality. Accordingly, the data requirements were encapsulated in the application, and even when programming languages began to acquire object-oriented facilities the more prominent role assigned to data was largely hidden from the users deep inside the application.

All of this seemed to work quite well until the need for interoperability and the attendant requirement for the exchange of data among applications surfaced. Two problems were quickly recognized. First, since each application controlled its own data schema the linking of multiple applications required application-to-application data mappings that led to hardwired systems. It soon became apparent that while it was possible to maintain the vertical flow of data within each of these stovepipe systems, it was inordinately difficult to exchange data horizontally between stovepipes. The second problem centered on this need for horizontal interoperability: How to exchange data between two stovepipe systems so that the receiving application will be able to process the imported data in a useful manner? There appeared to be two possible approaches for addressing this problem. To explicitly predefine the data exchange format and content, or to add meaning-identifiers to the data. The first approach, while providing a modest level of interoperability in the short term, exacerbated the problem in the long term. The hardwired data bridges were difficult and costly to maintain, provided little (if any) flexibility, and constituted multiple system failure points. The second approach led to the definition of standard data exchange protocols that conveyed to the receiving application at least some indication of the meaning of an imported data package. Of these protocols the Extensible Markup Language (XML) is rapidly gaining widespread acceptance. XML provides a degree of syntactic interoperability through nested data record delimiters (i.e., Unicode characters), data meaning-identifiers (i.e., tags), and links to other resources (i.e., Uniform Resource Identifiers).

Does a protocol like XML convey sufficient meaning to support horizontal interoperability? The answer is, no. The XML elements that are added to a data exchange package to convey meaning are of value only if the receiving application understands the name of each element. For example, the tag name "address" is only useful to the receiving application if it interprets that name to have the same meaning as the meaning assumed by the sending application (i.e., "address" could mean street address, e-mail address, object reference ID, etc.). However, XML does provide a syntactic foundation layer on which other layers such as the Resource Description Framework (RDF) can be built. The combination of these layers will serve as the enabling structure of what is referred to as the Semantic Web.

The vision of the Semantic Web is an information-centric environment in which autonomous software services with the ability to interpret data imported from other services are able to combine their abilities to accomplish some useful intent. This intent may range from simply finding a particular item of information to the more sophisticated tasks of discovering patterns of data changes, identifying and utilizing previously unknown resources, and providing intelligent decision-assistance in complex and time-critical problem situations. An example of such an environment is the TEGRID proof-of-concept system that was first demonstrated by the Collaborative Agent Design Research Center (CADRC) during a previous meeting of this Office of Naval Research Workshop/Conference series (Gollery and Pohl 2002). A brief summary of this demonstration is provided in the following section.

TEGRID: An Experimental Web Services System

The principal components of the TEGRID demonstration are ontology-based Web services that are capable of seeking and discovering existing Web services, extending their own information models through the information model of any discovered Web service, and automatically reasoning about the state of their internal information models. As shown in Figure 14, these components (referred to as Cyber-Spiders in TEGRID) consist of three principal components: a Web server; a semantic Web service; and, an information-centric application.

The Web server, utilizing the standard Hypertext Transfer Protocol (HTTP), serves as the gateway through which the Cyber-Spider gains access to other existing Web services. Existing Web servers primarily provide access to Hypertext Markup Language (HTML) data sources and perform only simple operations that enable access to externally programmed functionality. However, these simple operations currently form the building blocks of the World Wide Web.

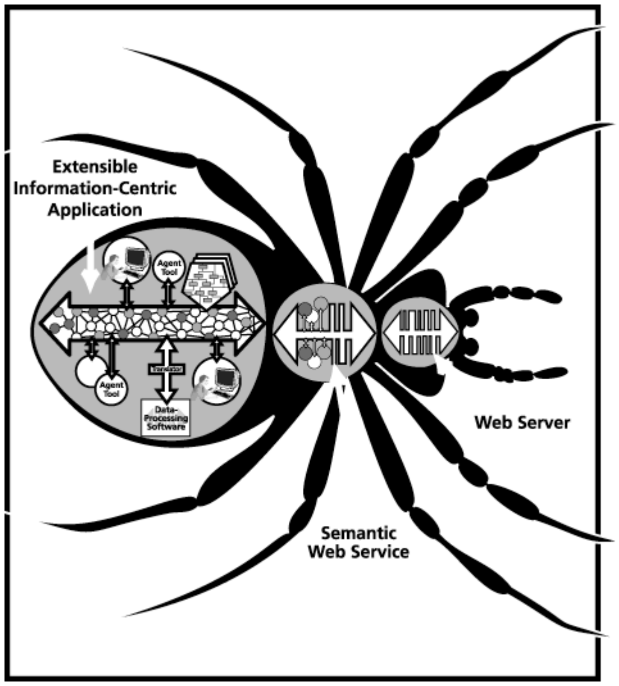
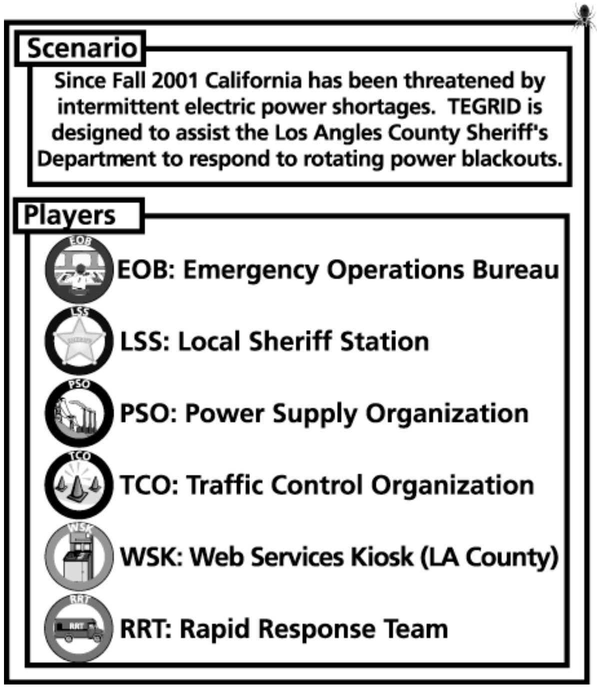


Figure14: Anatomy of a Cyber-Spider



Scenario

Since Fall 2001 California has been threatened by intermittent electric power shortages. TEGRID is designed to assist the Los Angles County Sheriff's Department to respond to rotating power blackouts.

Players

EOB: Emergency Operations Bureau
LSS: Local Sheriff Station
PSO: Power Supply Organization
TCO: Traffic Control Organization
WSK: Web Services Kiosk (LA County)
RRT: Rapid Response Team

Figure 15: Cast of TEGRID players

The second component of a Cyber-Spider is a semantic Web service (i.e., a Web service with an internal information model). A Web service is accessed through a Web server utilizing standard protocols (e.g., UDDI, SOAP, WSDL, SML) and is capable of providing programmed functionality. However, clients to a standard Web service are usually restricted to those services that implement specific predefined interfaces. The implementation of Web services in the Internet environment allows organizations to provide access to applications that accept and return complex objects. Web service standards also include a limited form of registration and discovery, which provide the ability to 'advertise' a set of services in such a way that prospective client programs can find services that meet their needs. The addition of an internal information model in a semantic Web service allows the storage of semantic level descriptions (i.e., information) and the performance of limited operations on these semantic descriptions. In other words, the semantic Web server component of a Cyber-Spider is capable of reasoning.

The third component of a Cyber-Spider is one or more information-centric applications. These applications are designed to take advantage of the resources provided by a number of semantic Web services, enabling them to reason about the usefulness of each service as a core capability within a more sophisticated set of discovery strategies. Moreover, the application component is able to construct relationships among the information models of different services, with the ability to integrate services without requiring agreement on a common information model.

With these three components Cyber-Spiders are at least minimally equipped to operate in an Internet environment as autonomous software entities, capable of: discovering needed services; accepting services from external offerers; providing services to external requesters; gaining context through an internal information model; automatically reasoning about available information; extending their information model during execution; extending their service capabilities during execution; and, learning from their collaborations.

The Cast of Players

Based on the scenario described in Figure 15, the TEGRID cast of players includes six semantic Web services: the Emergency Operations Bureau (EOB) of the Los Angeles Sheriff's Department; several Local Sheriff Stations (LSS); a Power Supply Organization (PSO); a Traffic Control Organization (TCO); several Rapid Response Teams (RRT); and, a Los Angeles County Web Services Kiosk (WSK).

Fundamental to each player are three notions. First, each player operates as an *autonomous* entity within an environment of other players. Most, but not all of the other players are also autonomous. This requires the autonomous players to be able to discover the capabilities of other players. Second, each autonomous player has a sense of *intent* to accomplish one or more objectives. Such objectives may range from the desire to achieve a goal (e.g., maintain situation awareness, coordinate the response to a time-critical situation, or undertake a predetermined course of action following the occurrence of a particular event) to the willingness to provide one or more services to other players. Third, each player (whether autonomous or not) is willing to at least *cooperate* with the other players. In some cases the level of cooperation will extend to a collaborative partnership in which the partnering players contribute to the accomplishment of a common objective. In other cases the cooperation may be limited to one player providing a service to another player, without any understanding or interest in the reason for the service request.

To operate successfully in such an autonomous Internet-based environment a Cyber-Spider player should be endowed with the following capabilities:

1. Subscribe to information from external sources (e.g., alerts, ontology extensions).
2. Accept subscriptions from external clients.
3. Dynamically change its subscription profile.
4. Extend its internal information representation.
5. Extend its own service capabilities.
6. Generate new agents for its own use.

7. Describe its own service capabilities to external clients.
8. Seek, evaluate and utilize services offered by external clients.
9. Provide services to external clients.
10. Describe its own (intent) nature to external clients.

The Cyber-Spiders in TEGRID are capable of demonstrating eight of these ten desirable capabilities. The ability of a Cyber-Spider to dynamically change its subscription profile, while technically a fairly simple matter, was not implemented because it is not used in the demonstration scenario. The ability of a Cyber-Spider to describe its own nature to external clients, on the other hand, is technically a much more difficult proposition. It will require a Cyber-Spider to have an understanding of its personality as a collective product of its internal information model and the relationship of that model with the external world. At best this must be considered a challenging research area that is beyond the current capabilities of information-centric software systems.

The Capabilities

The objective of the TEGRID scenario is to demonstrate the discovery, extensibility, collaboration, automatic reasoning, and tool creation capabilities of a distributed, just-in-time, self-configuring, collaborative multi-agent system in which a number of loosely coupled semantic Web Services associate opportunistically and cooperatively to collectively provide decision assistance in a crisis management situation. Specifically, these capabilities are defined as follows:

Discovery: Ability of an executing software entity to orient itself in a virtual cyberspace environment and discover other software services.

Extensibility: Ability of an executing software entity to extend its information model by gaining access to portions of the information model of another executing software entity.

Collaboration: Ability of several semantic Web Services to collaboratively assist each other and human users during time critical decision-making processes.

Reasoning: Ability of a software agent to automatically reason about events in near real-time under time critical conditions.

Tool Creation: Ability of a semantic Web Service to create an agent to perform specific situation monitoring and reporting functions.

The reasoning capabilities available in TEGRID are performed by software agents that are components of the players (i.e., the Cyber-Spiders). In other words, agents are predefined clients within player systems and perform internal functions that are necessary for the particular player to deliver its services and/or accomplish its intent. The following agents (i.e., collaborative tools) are available in the current TEGRID implementation:

Risk Agent: Assists the Emergency Operations Bureau to identify high-risk entities in the jurisdictional region of an activated Local Sheriff Station.

Deployment Agent: Assists the Emergency Operations Bureau to determine whether Rapid Response Team support is required for a particular activated Local Sheriff Station.

Power Level Agent: Assists the Power Supply Organization to determine if the electric power demand has exceeded supply.

Situation Agent: Assists the Emergency Operations Bureau to prepare and update its Status Report.

Station Monitor Agent: Assists the Emergency Operations Bureau to identify all Local Sheriff Stations that will experience power blackouts during the current and next blackout cycle.

Status Agent: Assists a Local Sheriff Station to prepare and update its Situation Status Report.

Local Station Agent: Assists a Local Sheriff Station to determine whether sufficient local resources are available to deal with current conditions.

Scheduling Agent: Assists the Emergency Operations Bureau to assign Rapid Response Teams and equipment to situations requiring their involvement.

Incident Agent: Assists the Emergency Operations Bureau to monitor the response to a particular situation supported by one or more of its Rapid Response Teams.

Routing Agent: Assists the Traffic Control Center to determine alternative routes to a particular situation location.

Demonstration Summary

Since the complete TEGRID demonstration scenario has been described elsewhere (Gollery and Pohl 2002) it will suffice here to summarize some typical events and automated reactions.

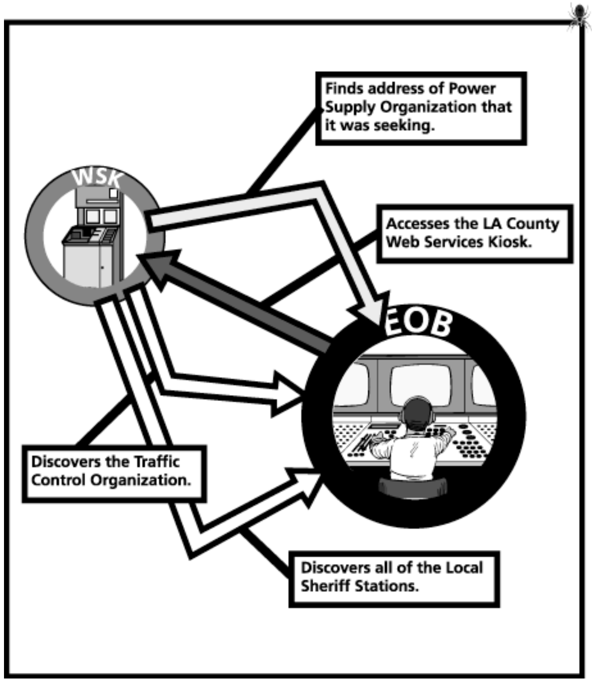

Finds address of Power Supply Organization that it was seeking.
WSK
Accesses the LA County Web Services Kiosk.
EOB
Discovers the Traffic Control Organization.
Discovers all of the Local Sheriff Stations.

Figure 16: Orientation and discovery

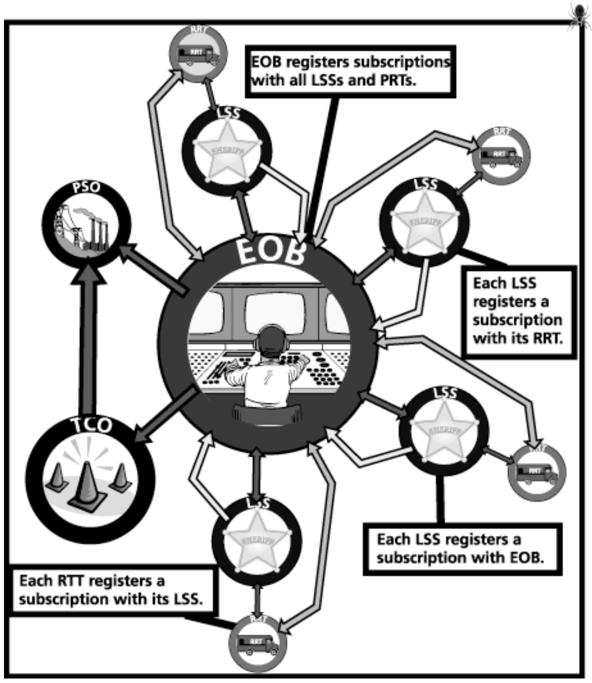

RRT
EOB registers subscriptions with all LSSs and PRTs.
LSS
RRT
PSO
LSS
EOB
Each LSS registers a subscription with its RRT.
TCO
LSS
RRT
LSS
Each LSS registers a subscription with EOB.
Each RTT registers a subscription with its LSS.
RRT

Figure 17: Information subscription

Orientation: The players orient themselves by accessing one or more directories of available services and registering an information subscription profile with those services that they believe to be related to their intent (Figure 16).

Subscription: The players access the services that they require to achieve their intent, register appropriate subscription profiles, and query for information that they believe to have a need for (Figure 17). For example, the Emergency Operations Bureau registers a subscription profile with each Local Sheriff Station, which includes all current police unit locations, mission completion events, new mission events, and any information changes relating to the availability of its Rapid Response Teams. Then queries each Local Sheriff Station for all information relating to its Rapid Response Teams and extends its information model. Finally, registers subscription profiles with each Rapid Response Team, the Power Supply Organization, and the Traffic Control Organization.

Collaboration: The Power Supply Organization first alerts its subscribers that a rolling power blackout condition is imminent (i.e., will commence per predefined schedule within 15 minutes) and subsequently alerts its subscribers that the rolling power blackout has commenced. The Emergency Operations Bureau (EOB) utilizes its Situation Agent to prepare the first version of the 'EOB Situation Status Report'. Then alerts all Local Sheriff Stations, in whose jurisdictions the next scheduled set of blackouts will occur, to prepare for potential deployment. And, finally, warns the Rapid Response Teams assigned to assist the Local Sheriff Stations in whose jurisdictions the next set of blackouts are scheduled to occur, to prepare for potential deployment. Consequently, all activated Local Sheriff Stations utilize their Status Agents to prepare the first version of their 'Situation Status Reports', the Local Sheriff Stations in whose jurisdiction the next set of blackouts is scheduled to occur, prepare for deployment.

Demonstration Results

The objectives of the TEGRID project were three-fold. First, to explore the primary capabilities that would be required of semantic Web services operating as largely autonomous decision-support components in a self-configuring, just-in-time, intelligent decision-assistance toolkit of collaborating software agents. Second, to determine if the currently available information-centric software technology could support at least basic (i.e., meaningful and useful) implementations of these required capabilities. And, third, to build a working experimental system that could serve as a test-bed for longer term research studies focused on the behavioral characteristics of self-configuring intelligent systems in general, and the ability of such systems to deal with specific kinds of dynamic and complex problem situations.

The demonstration showed that, today at a base level of functionality and in the near future at a much more sophisticated level, a Semantic Web environment will be able to support semantic Web services with the ability to: discover desired existing external services; accept and utilize services from external offerers; provide services to external requesters; gain understanding through the context provided by an internal information model; automatically reason about available information within the context of the internal information model; extend the internal information model during execution; spontaneously generate new agents during execution as the need for new capabilities arises; and, learn from the collaborations that occur within the cyberspace environment.

References

Berners-Lee T. and M. Fischetti (1999); 'Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by its Inventor'; Harper, San Francisco, California.

Cass S. (2004); 'A Fountain of Knowledge'; IEEE Spectrum (www.spectrum.ieee), Jan.30.

Dreyfuss H. (1979); 'What Computers Can't Do: The Limits of Artificial Intelligence'; Harper and Rowe, New York, New York.

Deyfuss H. and S. Dreyfuss (1986); ' Mind Over Machine: The Power of Human Intuitive Expertise in the Era of the Computer'; Free Press, New York, New York.

Dreyfuss H. (1997); 'What Computers Still Can't Do: A Critique of Artificial Reason'; MIT Press, Cambridge, Massachusetts.

Fellbaum C. (1998); 'WordNet, An Electronic Lexical Database'; MIT Press, Cambridge, Massachusetts.

Ganek A. and T. Corbi (2003); 'The Dawning of the Autonomic Computing Era'; IBM Systems Journal, 42(1) (pp.5-18).

Gollery S. and J. Pohl (2002); 'The TEGRID Semantic Web Application: A Demonstration System with Discovery, Reasoning and Learning Capabilities'; Office of Naval Research (ONR) Workshop Series on Collaborative Decision-Support Systems, hosted by the Collaborative Agent Design Research Center (CADRC) of Cal Poly (San Luis Obispo) in Quantico, VA, September 18-19.

Horn P. (2001); 'Autonomic Computing: IBM's Perspective on the State of Information Technology'; IBM Corporation, October 15 (www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf).

Lucas J. (1961); 'Minds, Machines and Goedel'; Philosophy, 36 (pp. 120-4).

Patterson D., A. Brown, P. Broadwell, G. Candea, M. Chen, J. Cutler, P. Enriquez, A. Fox, E. Kiziman, M. Merzbacher, D. Oppenheimer, N. Sastry, W. Tetzlaff, J. Traupman and N. Treuhaft (2002); 'Recovery-Oriented Computing (ROC): Motivation, Definition, Techniques, and Case Studies'; UC Berkeley, Computer Science Technical Report (UCB//CSD-02-1175), University of California, Berkeley, California, March 15, 2002.

Pedersen T. and R. Bruce (1998); 'Knowledge Lean Word-Sense Disambiguitization'; Proceedings 5th National Conference on Artificial Intelligence, July, Madison Wisconsin.

Pohl J. (1998); 'The Future of Computing: Cyberspace'; in Pohl J. (ed.) Advances in Collaborative Decision-Support Systems for Design, Planning, and Execution, focus symposium: International Conference on Systems Research, Informatics and Cybernetics, Baden-Baden, Germany, August 17-21 (pp.9-28).

Pohl J., A. Chapman, K. Pohl, J. Primrose and A. Wozniak (1999); 'Decision-Support Systems: Notions, Prototypes, and In-Use Applications'; Technical Report, CADRU-11-97, CAD Research Center, Design Institute, College of Architecture and Environmental Design, Cal Poly, San Luis Obispo, CA, January, 1997, reprinted July 1999 (pp.69-74).

Pohl J., M. Porczak, K.J. Pohl, R. Leighton, H. Assal, A. Davis, L. Vempati and A. Wood, and T. McVittie, and K. Houshmand (2001); 'IMMACCS: A Multi-Agent Decision-Support System'; Technical Report, CADRU-14-01, Collaborative Agent Design (CAD) Research Center, Cal Poly, San Luis Obispo, CA. (2nd Edition)

Random (1999); 'Random House Webster's College Dictionary'; Random House, New York, New York.

Searle J. (1980); 'Mind, Brains and Programs'; The Behavioral and Brain Sciences, 3 (pp. 417-24).

Searle J. (1992); 'The Rediscovery of the Mind'; MIT Press, Cambridge, Massachusetts.

Wooldridge M. and N. Jennings (1995); 'Intelligent Agents: Theory and Practice'; The Knowledge Engineering Review, 10(2) (pp.115-152).

Semantic Web Bibliography

Berners-Lee T. (2002); 'Weaving the Web'; Harper, San Francisco, California.

Berners-Lee T. (2004); 'What the Semantic Web Can Represent'; (www.w3.org/DesignIssues/RDFnot.html)

Berners-Lee T., J. Hendler and O. Lassila (2001); 'The Semantic Web'; Scientific American, May (www.scientificamerican.com/2001/0501issue/0501berners-lee.html)

Brickley D. and R. Guha (eds.) (2002); 'RDF Vocabulary Description Language 1.0: RDF Schema'; W3C Working Draft, April 30 (www.w3.org/TR/rdf-schema/)

Business Week (2002); 'The Web Weaver Looks Forward'; Interview with TIM Berners-Lee, March 27 (www.businessweek.com/bwdaily/dnflash/mar2002/nf20020327_4579.htm)

Carroll J. and J. De Roo (eds.) (2002); 'Web Ontology Language (OWL) Test Cases'; W3C Working Draft, October 24 (www.w3.org/TR/2002/WD-owl-test-20021024/)

Casey M. and M. Austin (2001); 'Semantic Web Methodologies for Spatial Decision Support'; Institute for Systems Research and Department of Civil and Environmental Engineering, University of Maryland, November.

Cohen P., R. Schrag, E. Jones, A. Pease, A. Lin, B. Starr, D. Easter, D. Gunning and M. Burke (1998); 'The DARPA High Performance Knowledge Bases Project'; Artificial Intelligence Magazine 19(4) (pp.25-49) (reliant.teknowledge.com/HPKB/Publications/AImag.pdf)

DAML-ONT (2000); (www.daml.org/2000/10/daml-ont.html)

DAML+OIL (2001); (www.daml.org/2001/03/reference.html)

DAML-S (2002); (www.daml.org/services/daml-s/0.7/)

Dean M., D. Connolly, F. van Harmelen, J. Hendler, I. Horrocks, D. McGuinness, P. Patel-Schneider, F. Stein and L. Stein ((eds.) (2002); 'OWL Web Ontology Language 1.0 Reference'; W3C Working Draft 29, July and November 12 (www.w3.org/TR/owl-ref/)

Daconta M., L. Obrst and K. Smith (2003); 'The Semantic Web: A Guide to the Future of XML, Web Services, and Knowledge Management'; Wiley, Indianapolis, Indiana.

Ewalt D. (2002); 'The Next Web'; Information Week, October (www.informationweek.com/story/IWK20021010S0016)

Fikes R. and D. McGuinness (2001); 'An Axiomatic Semantics for RDF, RDF Schema and DAML+OIL'; KSL Technical Report (KSL-01-01), October (www.ksl.stanford.edu/people/dlm/daml-semantics/abstract-axiomatic-semantics.html)

Garshol L. and G. Moore (eds.) (2002); 'The XML Topic Maps (XTM) Syntax'; JTC1/SC34:ISO 13250, July 22 (www.y12.doe.gov/sgml/sc34/document/0328.htm)

Gil Y. and V. Ratnakar (2002); 'Markup Languages: Comparison and Examples'; Information Sciences Institute, University of Southern California, TRELLIS project

(www.isi.edu/expect/web/semanticweb/comparison.html)

Heflin J., R. Volz and J. Dale (eds.) (2002); 'Requirements for a Web Ontology Language'; W3C Working Draft, July 8 (www.w3.org/TR/webont-req)

Hendler J., T. Berners-Lee and E. Miller (2002); 'Integrating Applications on the Semantic Web'; Journal of the Institute of Electrical Engineers of Japan, 122(10), October (pp.676-680).

Horrocks I. (2002); 'DAML+OIL: A Description Language for the Semantic Web'; IEEE Intelligent Systems, Trends and Controversies.

Manola F. and E. Miller (eds.) (2002); 'RDF Primer'; W3C Working Draft, March 19 (www.w3.org/TR/2002/WD-rdf-primer-20020319/)

OIL (2004); (www.ontoknowledge.org/oil/)

Ontolingua (2004); (www.ksl.stanford.edu/software/ontolingua/)

OWL (2001); 'The Web Ontology Language'; (www.w3.org/2001/sw/WebOnt/)

Patel-Schneider P., I. Horrocks, P. Payes and F, van Harmelen (eds.) (2002); 'Web Ontology Language (OWL) Abstract Syntax and Semantics'; W3C Working Draft, November 8 (www.w3.org/TR/2002/WD-owl-semantics-20021108/)

Swartz A. (2002); 'The Semantic Web in Breadth'; (logicerror.com/semanticWeb-long)

W3C (1999); 'Resource Description Framework (RDF) Model and Syntax Specification'; W3C Recommendation, February 22.

W3C (2001); 'XML Linking Language (Xlink) Version 1.0'; W3C Recommendation, June 27 (www.w3.org/TR/xlink/)

W3C (2003); 'Design Issues'; (www.w3.org/DesignIssues/diagrams/sw-stack-2002.png)