22nd International Conference on: Systems Research, Infomatics and Cybernetics

Preconference Proceedings

Advances in Adaptive Planning Capabilities

Focus Symposium: Baden-Baden, Germany; Tuesday, August 3, 2010

Focus Symposium Chair

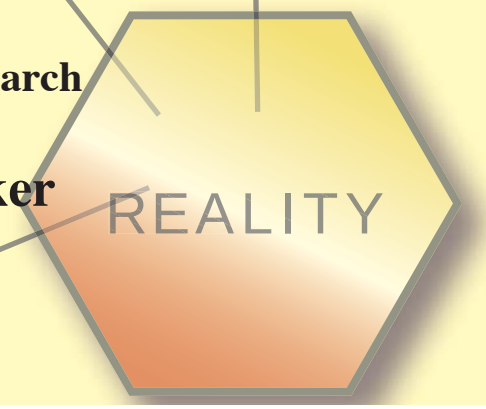
Jens Pohl

Executive Director, Collaborative Agent Design Research Center
and Professor of Architecture
California Polytechnic State University
San Luis Obispo, California, USA

Sponsored by:

The International Institute for Advanced Studies
in Systems Research and Cybernetics
and
Society for Applied Systems Research

Professor George E. Lasker
Chairman



ISBN 978-1-897233-17-7

InterSymp – 2010

22nd International Conference on Systems Research, Informatics and Cybernetics

(August 2-6, 2010, Baden-Baden, Germany)

Pre-Conference Proceedings of the Focus Symposium on

Advances in Adaptive Planning Capabilities

Tuesday, August 3, 2010

Focus Symposium Chair:

Jens Pohl
Executive Director, Collaborative Agent Design Research Center
and Professor of Architecture
College of Architecture and Environmental Design
California Polytechnic State University
San Luis Obispo, California, USA

Sponsored by:

The International Institute for Advanced Studies
in Systems Research and Cybernetics
and
Society for Applied Systems Research

Professor George E. Lasker
Chairman

ISBN: 978-1-897233-17-7

Preface

The theme of this focus symposium, *Advances in Adaptive Planning Capabilities*, draws attention to the rapid changes that are impacting our daily business, recreational and personal endeavors. The pace of our daily activities appears to be increasing at an accelerating rate as we are increasingly taking advantage of the benefits of global connectivity and powerful computer-based tools. These are of course the principal drivers of the Information Age. Yet, a case can be made that we are really only at the threshold of the Information Age and that we are therefore experiencing only a very small dose of the changes that the Information Age has in stall for us.

A painful lesson that we are currently learning is that there is a fundamental difference between data and information. One of the advances in computer hardware over the past years has been the availability of very small, but high volume, mass storage devices with access and data retrieval speeds measured in milliseconds. One can purchase such a device (i.e., disk drives) with a storage capacity of two terabytes at a local electronics shop for less than (US) $200 at the time of this InterSymp-2010 Conference. This is an enormous storage capacity when one considers that all of the books and documents in the United States Library of Congress would require less than 12 terabytes of storage.

The ability to store such vast amounts of data has multiple impacts that produce human tension, by forcing us to acknowledge the difference between data and information. The principal purpose for storing data is for us to use the data to make better decisions and formulate sounder plans based on more accurate and comprehensive information. However, therein lays the problem. Data are just numbers and words that have meaning to us simply as symbols (e.g., house, fish, 16, and so on), but are of real value only when they are used in a particular context. Within this context the numbers and words become information that can be used for planning purposes and to make decisions during the execution of a course of action. If computers cannot provide the appropriate context with the vast amount of data that they are able to store then it is left to the human computer users to interpret the data within the context held in their cognitive facilities. As a result the human user is overwhelmed by the amount of data that needs to be interpreted and becomes a bottleneck within the information management continuum. One could therefore argue that we are still entrenched in the Data Age and only gradually transitioning into the Information Age.

In what way is this difference between data and information related to *adaptive planning*? We must first ask: What is the difference between *planning* and *adaptive planning*? The concept of *adaptive planning* recognizes that even the most diligently formulated plans are seldom executed without the occurrence of some event that requires the plan to be changed. As our expectations of efficiency, guaranteed timely delivery, and minimum wastage of material increase, the ability to adapt to potentially disruptive situations occurring during execution gains critical importance. The business objectives of *end-to-end supply chain management* and *just in time inventory* are particularly vulnerable to unexpected events that can cause major disruptions. Therefore the need for *adaptive planning capabilities* arose initially in the business and military domains where resource and result sensitive processes had to be accomplished under time-critical conditions. To satisfy this emerging need the emphasis has increasingly shifted in recent years from planning tools to re-planning tools. It is also being recognized that to reduce the risks involved in *end-to-end supply chain management* it is not sufficient to be able to re-plan after an event has occurred during execution (i.e., reactively), but it is also necessary to be able to foresee potential

disruptive events and re-plan proactively. Clearly, this requires the computer to be able to continuously monitor the execution process, analyze the information collected, identify even subtle trends that may not be obvious to the human user, generate warnings and alerts whenever the detected trends may lead to disruptions, and generate alternative plans. To accomplish this in an automatic fashion the computer will require software that is able to interpret data within the appropriate context and then reason about the information that it has generated.

From a broader point of view the concept of *adaptive planning* applies not only to supply chain management processes, but to virtually all decision-making domains in which there exists a high degree of complexity. Typically, such domains share three characteristics. First, there are many factors that need to be considered. Second, there are many relationships that make any one factor dependent on several other factors and sometimes most of the other factors. It is these relationships that contribute greatly to the complexity of the domain. Third, there exists a degree of uncertainty because not all of the necessary information is available, requiring decisions to be made on the basis of incomplete information. Therefore, it is not surprising that the papers that are included in these Symposium Proceedings cover several domains ranging from network complexity and cyber security to architectural design, natural language processing and context representation.

All of this points to the urgent need for intelligent software that has some understanding of the data being processed and is therefore able to collaborate in a meaningful manner with the human computer user. The fundamental driver of this need is the human expectation that in the so called Information Age the computer will function as a partner to the human user with the capability to automatically interpret the meaning of data in the applicable context, proactively identify events and trends, monitor execution sequences, and reactively generate modified plans.

Taking architectural design as an example, since two of the Focus Symposium papers address this domain[1], the need for intelligent software with *adaptive planning* capabilities has become more pronounced with the increased concerns for sustainability of the built environment within the fragile ecological balances of the natural environment. The overarching impact of these ecological concerns is that the design of buildings will become an increasingly more complex undertaking.

Whereas architects practicing in the 20th Century already had to deal with a host of often conflicting design issues ranging from space planning and three-dimensional modeling to structural and environmental system selection, 21st Century architects will have many more considerations added to their plate. For example, they will need to justify the use of every material, not only in respect to cost and serviceability but also based on embodied energy and potential toxicity parameters, as well as the ability to recycle the material. The need to minimize water usage will require the use of graywater with the necessary capture and recycling facilities. Most, if not all, of the energy used in a new residential building will most likely have to be captured on-site. Under these circumstances, the achievement of a building design that would have been acclaimed in the 1990s as an award winning energy conscious scheme might become a barely baseline solution in the not too distant future.

[1] See *Knowledge-Based Collaborative Architectural Design* by Carrara, Fioravanti and Nanni from the Sapienza University of Rome in Rome, Italy; and, *Intelligent Software for Ecological Building Design* by Pohl J., Assal and Pohl K. from the California Polytechnic State University and CDM Technologies, Inc. both in San Luis Obispo, California, USA.

Based on current and historical building construction and occupancy experience it is quite difficult to imagine the design and operation of a building that is not in some measure destructive to the natural environment. Typically: the site is graded to provide convenient vehicular access and suit the layout of the building and its immediate surroundings; the construction materials and components are produced from raw materials that are extracted from nature and consume a great deal of energy during their production; the materials and components are transported to the site consuming more energy in transit; on-site construction generates waste in terms of packaging material and the fabrication of footings, walls, floors, and roof; during the life span of the building, energy is continuously consumed to maintain the internal spaces at a comfortable level and power the multiple appliances (e.g., lights, communication and entertainment devices, food preservation and preparation facilities, and security systems); despite some concerted recycling efforts, much of the liquid and solid waste that is produced during the occupancy of the building is normally collected and either treated before discharge into nature or directly buried in landfills; and finally, at the end of the life span when the building is demolished most, if not all, of the construction materials and finishes are again buried in landfill sites.

Let us consider the other extreme, a building that has been designed on ecological principles and is operated as a largely self-sufficient micro-environment. Ecological design has been defined in broad terms as being in symbiotic harmony with nature. This means that the building should integrate with nature in a manner that is compatible with the characteristics of natural ecosystems. In particular, it should be harmless to nature in its construction, utilization, and eventual demolition. The closest we come to being forced to comply with such stringent design and occupancy requirements is in the realm of extraterrestrial habitats, such as an outpost on the Moon or Mars. The constraints imposed by the severe transportation limitations and the hostility of the environment to human, animal, and plant life, require careful consideration of even the smallest component or quantity of material and the most minute energy requirement and need for non-recyclable material. The designers of such extraterrestrial buildings will be faced with design criteria that are only slightly more stringent than those called for by a truly ecological design on Earth. For example:

- In the absence of any excavation equipment the footings of the building will need to adjust to the site topography, rather than the converse. Under these circumstances careful site selection will be a necessary prerequisite to any successful construction project. Also, to accommodate changes in topography that could occur due to environmental influences, the footings will need to be adjustable at least in height. While ecological design on Earth may tolerate a slightly larger building footprint, any significant reshaping of the site topography and certainly larger areas covered by building footings or paving should be avoided.
- The building will need to be designed as a minimum weight structure, since every pound of material would need to be transported from Earth with an enormous consumption of energy. While the use of on-site materials would circumvent the transportation problem, this alternative is unlikely to be a feasible option at least during the early stages of extraterrestrial settlement due to the absence of the necessary extraction and manufacturing facilities. The adoption of minimum weight structural principles on Earth could also be a desirable ecological design criterion. It would serve to minimize the size of footings, reduce the consumption

of energy required for transporting materials and components to the site, and require fewer raw materials to be mined from the Earth's surface.

- The building will need to be largely self-sufficient in terms of the energy required to sustain its occupants. This includes environmental control (i.e., temperature, humidity, air quality, and air composition), food preservation and preparation equipment, water and other waste recycling systems, communication and computer hardware devices, and any other electronic monitoring and control facilities. With the exception of the need to maintain an artificial atmosphere (i.e., air composition) within the building, these requirements are essentially the same as those prescribed by ecological design principles on Earth.
- The occupants of the extraterrestrial building will depend on the treatment and reuse of graywater and the recycling of solid waste to virtually the same extent as a building on Earth that adheres strictly to ecological design principles. In both cases water emerges as one of the most precious and essential resources for the sustainment of human life.
- Apart from the treatment and reuse of graywater, the building will need to incorporate a waste management system that is capable of sorting dry waste as a precursor to recycling and processing wet waste in an anaerobic or similar treatment facility for composting purposes.
- Regardless of whether the building is intended for an extraterrestrial or terrestrial location, it should be designed for a fixed life span. The building materials and component products will need to be reusable in some form at the end of that life span. To satisfy this ecological design requirement the building must be deconstructable, the materials must be recyclable, the products must be disassemblable, and the materials dissipated from recycling must be harmless. The concept of a *closed-loop* building material strategy is central to ecological design and *green building* principles.

This is a complex undertaking requiring the human designer to have both breadth and depth of knowledge. Designers will not be able to deliver this level of expertise based on intuition, experience, or manual methods alone. Instead, they will require sophisticated simulation tools that are user-transparent and seamlessly integrated as semi-automated services within the kind of intelligent computer-aided design environment that can be made possible by a distributed Web-enabled environment based on service-oriented architecture principles as described in one of the papers in these Proceedings[2].

Jens Pohl, June 2010

(jpohl@calpoly.edu) (www.cadrc.calpoly.edu)

[2] *On the Road to Intelligent Web Applications* by Assal and Pohl K. from the California Polytechnic State University and CDM Technologies, Inc. both in San Luis Obispo, California, USA.

InterSymp-2010
International Conference on Systems Research, Informatics and Cybernetics

Focus Symposium
on
Advances in Adaptive Planning Capabilities

Tuesday, August 3, 2010

TABLE OF CONTENTS

Solving the Data Deluge Problem

Jens Pohl, Ph.D.

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

Abstract

The paper postulates that the information technology revolution that is commonly referred to as the Information Age is currently in a transition stage between data-processing and knowledge management that should be more aptly referred to as the Data Age. Symptoms of this transition stage are a data deluge problem that is evidenced by the inability of human computer-users to effectively analyze and draw useful conclusions from the overwhelming volume of data that is being collected, the increasing complexity of networked systems, and the acknowledged vulnerability of virtually all existing digital systems to cyber security threats.

The author suggests that the core cause of the data deluge problem is that existing computer software systems are largely confined to the processing of atomic data elements rather than meaningful information. With the incorporation of a virtual model of the relevant real world knowledge domain it is possible for computer software to interpret the meaning of data within the context provided by the model. Such models can be constructed in the form of an ontology that is machine processable and accessible to inferencing modules referred to as agents. Context-based information-centric software provides a level of artificial intelligence that can be effectively used to mitigate the current data bottleneck, to shield the human user from the technical complexities of distributed systems, and to maintain an acceptable level of cyber security.

Keywords

Agents, autonomic computing, context, cyber security, data, Data Age, data-centric, information, Information Age, information assurance, information-centric, intelligence, networks, ontology, representation, security,

1. Introduction

When we use search engines to find some information on the Internet, we typically receive more links to potential information sources (i.e., hits) than we care to look at. We have learned from experience that many of the hits will be disappointing because they do not lead to the information that we are seeking. Soon after the terrorist attacks on the United States in September 2001, much evidence was found that several warnings of a planned attack were contained in the routinely collected intelligence data, but had been overlooked. The military are so inundated with sensor data from satellites, unmanned aerial vehicles, and land-based sources that they cannot possibly analyze in near-real time. These are all symptoms of a rapidly escalating data deluge problem.

The amount of data that is being collected by our global digital infrastructure far exceeds our human ability to interpret, analyze, draw conclusions, and act upon under even less than time-critical circumstances. This is not a problem that occurs only under special circumstances, such as the relief operations after a major national disaster. Rather, the data deluge problem is clearly becoming more pronounced. The reason is quite simple; - while the volume of data is increasing exponentially our human ability to interpret the data is increasing linearly. Clearly, if we continue to apply the same methods to this problem then we are destined to fall further and further behind.

So, what is at the core of this problem? Is it that we are collecting more data than we need? Or, perhaps, faster computers and better collection methods will eventually overcome the problem. No, we must not limit our data collection capabilities and even faster computers are not going to solve the problem because the volume of data is increasing at an even faster rate. Instead we must find a way of automating the interpretation and analysis of data. Although we have proclaimed for some time to have entered the Information Age we are still largely immersed in what might be more appropriately referred to as the Data Age.

2. Difference between Data and Information

It is a common misconception that with the entry into the Information Age we are overwhelmed by an overabundance of information. A more accurate characterization would be that while we are being subjected to a deluge of data we are in fact typically faced with a scarcity of information. There is a major difference between data and information. Data are simply numbers and words such as *rain*, *traffic*, *4*, *mph*, *car*, *police*, *intersection*, *184*, *Grand Junction*, *100*, *bridge*, *crossing*, and so on (Figure 1). These are all symbols that we understand and are able to reason about when they appear in some *context* such as the following sentence: "*... police car 64 crossing Grand Junction bridge at 100 mph.*"

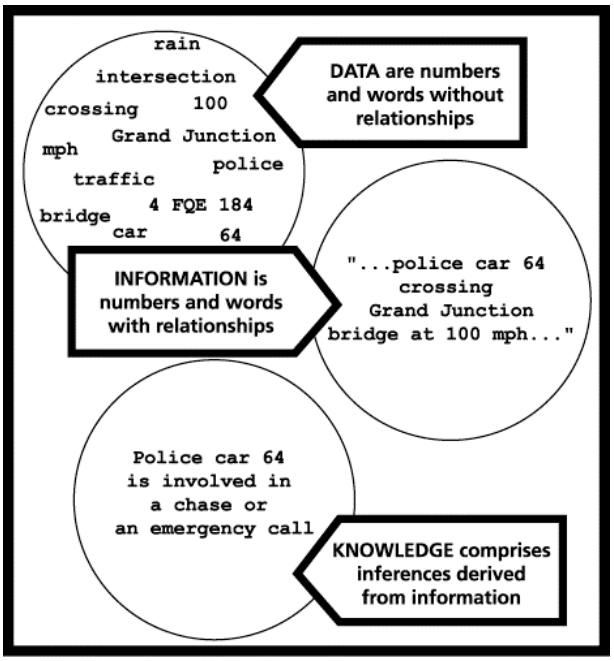


Figure 1: Definition of terms

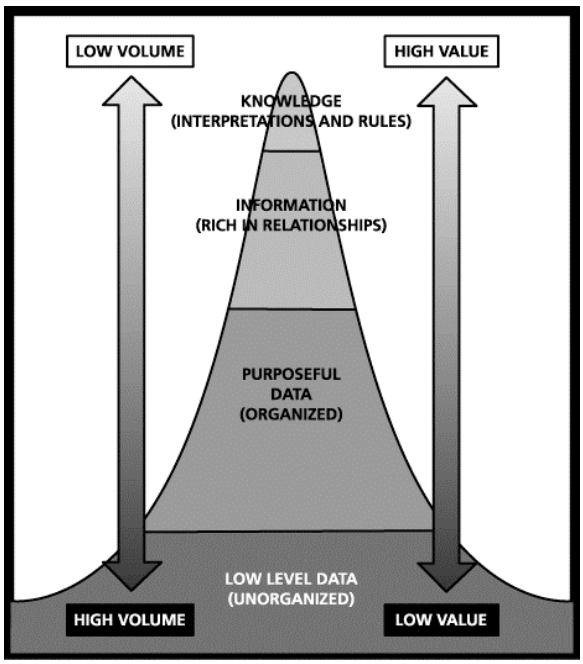


Figure 2: Transition from data to knowledge

It is not difficult for us to infer from the *context* of this sentence that if a *police car* is traveling at *100 mph* over a *bridge* in what is presumably a populated *Grand Junction* neighborhood then it is likely to be involved in a chase or emergency call. We are able to interpret data into information by utilizing the *context* that we have accumulated in our brain over time (i.e., our experience). At the lowest structural level this *context* consists of the implied relationships that bind the words and numbers together into a meaningful piece of information. For example, in our brain the symbol *bridge* is associated with characteristics that relate to the purpose of a bridge and how such a purpose can be achieved to the best of our engineering knowledge, and how this engineering problem has been solved in the past based on our life experience. Based on structural engineering principles a bridge has to span over some horizontal distance and is therefore likely to be limited in width due to structural constraints and cost. This relates well to our past experience of driving cautiously over bridges that are typically narrow, elevated, and subject to a speed limit.

Computers were invented in the 1940s because there was an urgent need for a device that could compute numbers much faster than a human mathematician. Applications that required this superhuman computational speed were principally related to transportation (i.e., navigational charts) and warfare (i.e., artillery tables). Much later in the 1970s it became apparent that the ability of computers to process large volumes of data was a very useful and potentially even more important capability. With the miniaturization of electronic components both the power and data storage capacity of computers has increased over the past two decades by factors of six and four every three years, respectively. Today (2010) an electronic mass storage device (i.e., disk drive) with a storage capacity of two terabytes (i.e., two thousand billion bytes) can be purchased for less than $250 at a local retail store. The enormity of that data storage capability becomes clear when we consider that the millions of books and documents in the Library of Congress collection will require a data storage capacity of less than 12 terabytes.

The principal reason for storing data is for analysis, for monitoring trends, and for planning purposes. However, these data-processing tasks must be performed in consideration of the *context* in which the data have been generated. While the quantity of data stored in computers was still relatively small these tasks could be performed by the human users who provided the necessary *context* by drawing on the experience and knowledge stored in their brain. Today, the volume of data that is stored in computers far outstrips the ability of the human user to interpret, analyze, and detect any but the most obvious trends.

3. The Need for Software Intelligence

Could the computer become an extension of the *context*-based data interpretation and analysis capabilities of the human user? This would require some representation of *context* to be embedded in the software components of a computer-based environment. Data in *context* is analogous to information, which by definition represents meaning. Therefore, software that is capable of processing information, commonly referred to as information-centric software to distinguish it from data-centric software, has by implication some degree of *understanding* of the data that it is designed to process. With even a relatively limited capacity to *understand* the meaning of data it becomes possible to incorporate in the software automated data interpretation, analysis and trend detection capabilities that may be characterized as *intelligent* features.

There are at least three compelling reasons why software that is enabled with the ability to process data within the *context* that the data are relevant is an essential prerequisites for exploiting the opportunities and combating the security risks posed by the Information Age.

Reason (1) – Increasing Data Volume: The first reason relates to the current data-processing bottleneck. As mentioned previously, advancements in computer hardware 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 processing 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 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.

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 2). 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. 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.

Reason (2) – Increasing Network Complexity: 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 and security of such information technology environments as the key enabler of their effectiveness, profitability and continued existence. 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, 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, 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.

Clearly, 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 the software (i.e., the automatic interpretation of information by the computer).

Reason (3) – Increasing Cyber Security Vulnerability: The third reason is related to cyber security and, in particular the prevention of unauthorized network intrusions and the protection of the data that are transmitted and stored within networks. Traditionally, the security of computer-based systems has relied on the physical separation of classified and non-classified systems and data encryption. Certainly, physical separation is neither desirable nor viable in a society that increasingly demands and depends on near-instant global connectivity. Yet, at the same time, we must ensure that the cross-domain transfer of data will be accomplished transparently, seamlessly, and securely.

Under these circumstances it can be assumed that information assurance must increasingly rely on measures that focus on securing the data within the network, on the assumption that the network can be penetrated by a skillful intruder (CBS 2009, CNN 2009). This should not at all imply that network access protection is considered to be ineffective and therefore unimportant. However, past experience has shown that even our apparently most secure military, intelligence, and commercial networks can be and have been penetrated. It is unlikely that implementation of the most sophisticated multi-biometric authentication measures cannot be *spoofed* (i.e., fooled through impersonation). Therefore, protection of the data that reside in our networks represents the final and most critical level of cyber defense.

Clearly, absolute information assurance cannot be achieved. There will always be someone who will devise an ingenious method for circumventing whatever security measures have been implemented. This may involve the development of new technology, such as the Colossus

machine that successfully deciphered the German Enigma code during the Second World War, or an isolated human lapse that can jeopardize the most stringently controlled security precautions. The best that we can aim for is an *acceptable* level of security that includes multiple gateways and, in particular, can detect network intrusion and compromised data at the earliest instance.

While the encryption of data is an essential security precaution, it is by itself not sufficient to provide an *acceptable* level of security. There is a need for software that is able to categorize that data that are processed and stored within a network into degrees of sensitivity and apply multiple levels of security depending on the nature of the data. Technology that is able to identify and extract secret data elements from a data stream, encrypt them, and store them in a dispersed manner across multiple networks, is already available even though it may not yet be employed in our networks. At the very least such data security software should be immediately evaluated with a view to accelerated implementation.

However, even this level of security will not provide an *acceptable* level of information assurance in the near future, nor does it exploit the full capabilities of our current IT knowledge. There is an urgent need to apply information-centric concepts and software design principles to protect the data in our networks. In this respect information-centric refers to the implementation of data security methodologies that are based on an understanding of the content (i.e., meaning) of data and the *context* within which the data are intended to be used, or could be misused by a rogue party. The ability to apply information assurance technologies that are capable of automatically imposing multi-level data security measures based on the ability of software to have some understanding of the meaning and *context* of a data stream is in the opinion of this author a fundamental prerequisite for the achievement of an *acceptable* level of information assurance.

4. Information-Centric Software: *An Urgent Need and Unique Opportunity*

Few will argue that computer-based systems will become increasingly more intelligent in the near future. The signs of this have already become apparent in fraud detection software employed on a daily basis by the insurance, banking and communication industries, by the adaptive decision-support systems used for military planning and re-planning, and by the findings that are being published in the research community. The ability of computer software to automatically extract meaning from unstructured text through the automated interpretation of data within the applicable *context* represents a paradigm shift in human endeavors. It will become a principal differentiator between competitiveness and non-competitiveness in the global marketplace and between *acceptable* and unacceptable levels of information assurance in the homeland security and national defense arenas. Our Government has an urgent need to accelerate this paradigm shift through leadership and, at the same time, take advantage of a unique opportunity to decisively enable our industrial complex in an increasingly competitive world-wide marketplace.

So far the US Government appears to have taken a decidedly reactive stance in respect to both cyber security and advances in information management. It has relied largely on industry to lead the way and seen neither the necessity nor the opportunity of implementing a concerted effort of appropriate proportions to ensure that the US will regain the preeminent position that it held in IT some 20 years ago. Over the past two decades our leadership has eroded to the point where

our networks are subjected to daily intrusion (much of it probably unknown to us) and our most secret data assets are not secure.

5. Context Representation and Intelligent Tools

The ability to represent *context* in computer software has been available for at least the past 30 years (Winston 1970, Biermann and Feldman 1972, Cohen and Sammut 1978). Hampered initially by a lack of hardware power and later by the absence of any compelling need to involve the computer in the direct interpretation of data, these information modeling techniques were not applied in the mainstream of computer software development until fairly recently. The compelling reasons that have suddenly brought them to the foreground are the increasing volume of computer-based data that is beginning to overwhelm human users, and the homeland security concerns that emerged after the tragic September 11, 2001 terrorist incidents in the United States.

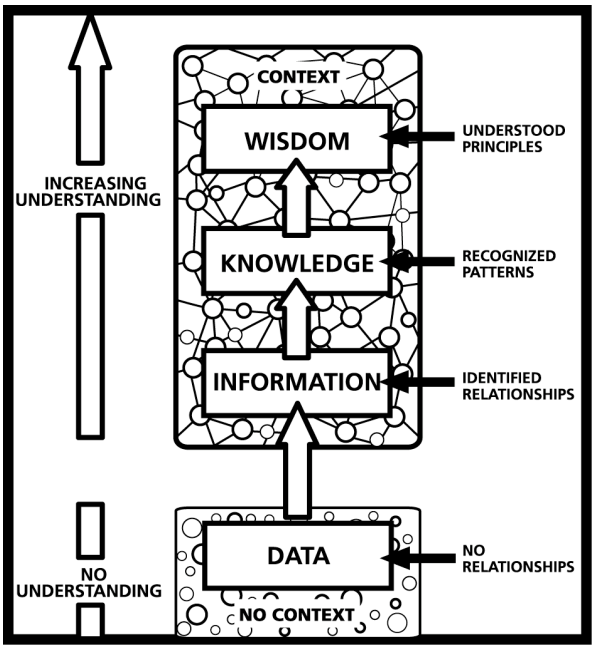


Figure 3: The paradigm shift from data to information with relationships

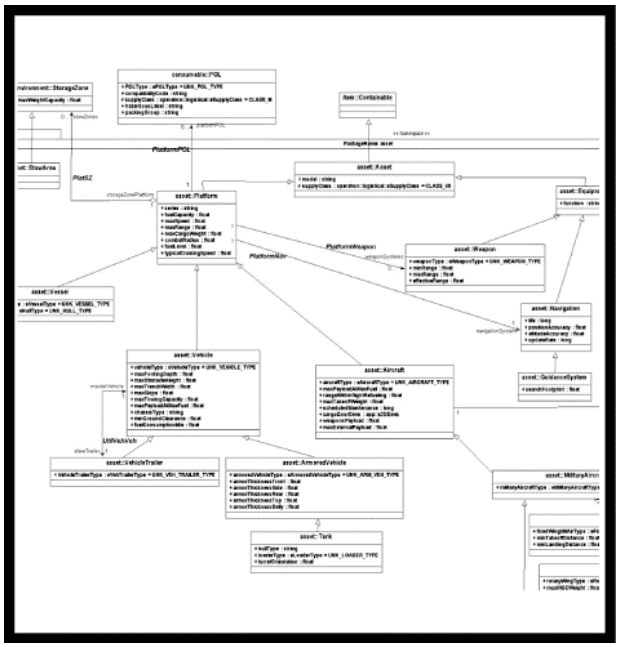


Figure 4: Portion of a typical information model (ontology) in the logistic domain

The physical gap that is shown schematically between the realms of the data environment without context and no understanding, and the information environment with context and ascending levels of greater understanding in Figure 3, underscores the fundamental difference between the two realms. The transition from data-processing software to information-centric software requires a paradigm shift in the human perception of the role of computers. By incorporating an internal information model (i.e., ontology) that represents portions of real world context as a virtual environment of objects their characteristics and the associations that relate these objects, information-centric software is capable of performing a useful level of automatic reasoning. A number of software agents with relatively simple reasoning capabilities are able to collaborate and through their collective efforts come to more sophisticated conclusions.

For the computer to be able to support automatic reasoning capabilities we have to create a software environment that incorporates context. This can be achieved fairly easily by

constructing an information model as a virtual representation of the real world context within which software agents are expected to apply their reasoning capabilities. Such an internal information model is referred to as an ontology. A small part of a typical example of such an ontology is shown in Figure 4. It describes the real world context in terms of objects with characteristics and relationships. For example, in a military command and control context such objects would include different kinds of weapons, a wide range of infrastructure objects, weather forecasts, friendly and enemy units, and even conceptual objects such as the notions of threat, planning, mobility, and readiness. Generally speaking, the more relationships among objects that are included in the ontology the more context is provided by the ontology, and the more powerful (i.e., intelligent) the reasoning capabilities of the software agents are likely to be.

Without the context provided by an internal information model (i.e., ontology) there can be no meaningful, automatic reasoning by software agents. Therefore, the essential prerequisite for intelligent agents is the existence of an internal information model that provides the necessary context for the symbolic reasoning activities of the agents. We human beings do not have to consciously invoke any action to relate what we see, hear and feel to the context held in our brain. The need for this context to be created in the computer is therefore not intuitively obvious to us. This is no doubt the principal reason why such a fundamental aspect of intelligent computer-based agents is still largely overlooked.

Software Agents – Automated Reasoning: There are several types of software agents, ranging from those that emulate symbolic reasoning by processing rules, to highly mathematical pattern matching neural networks, genetic algorithms, and particle swarm optimization techniques. The focus here is on ontology-based software that utilizes agents with symbolic reasoning capabilities. These agents may be described as software modules that are capable of reasoning about events (i.e., changes in data received from external sources or as the result of internal activities) within the context of the information contained in the internal information model (i.e., ontology). Since such agents are also often loosely referred to as intelligent agents the question arises whether computer intelligence is really possible? From a commonsense point of view it would appear that humans have intelligence and computers are just very fast but unintelligent machines. Looking at this question from an entirely human point of view we may well come to such a conclusion.

Human intelligence is only one kind of intelligence that is strongly influenced by the biochemical nature of the human body and its cognitive facilities, while the artificial intelligence that can be embedded in computer software is another kind of intelligence altogether. To differentiate human intelligence from the seemingly intelligent capabilities of computers we need to entertain the notion that there are levels of intelligent behavior. Remembering is probably the lowest level of intelligence. Certainly computers can store vast amounts of data and can retrieve these data quickly and accurately. However, from our human point of view remembering is more than just retrieving data. Remembering also involves relationships and context, which makes data meaningful and relevant. Therefore, by including an information model (i.e., ontology) in a software application or service we are able to represent information rather than data in the computer. This allows us to include modules in the software (i.e., software agents) that are able to automatically reason and communicate the results of their reasoning activities to other agents, including human users. We are creating in this way a virtual copy of the context of a problem situation in the computer-based environment. The players (i.e., the agents) in this virtual environment can assume many different roles and can contribute and collaborate at many levels,

ranging from the categorization of data to the identification of patterns and the recognition of relationships that may have been overlooked by human users.

In this way, if we store not only data in the computer but also the relationships that convert such numbers and words into information then we can also embed in the software rule sequences that are capable of reasoning about this information. Such sequences may be as simple as condition-action statements. For example, *if* an enemy tank unit is sighted *then* place a call-for-fire on the enemy tank unit *and* commence the process of weapon selection. However, the same software might also incorporate agents that perform more sophisticated tasks. For example, selecting the best mix of lift assets (e.g., helicopters, hovercraft, vertical take-off aircraft, etc.) to transport a wide range of supplies to multiple landing zones within requested time windows, and within constraints such as weather conditions, enemy actions, and so on. The latter agents consider results received from other agents, and utilize a wide range of heuristic and algorithmic methods to arrive at a possible solution.

Ontology – Context Representation: How can we embed in software a virtual version of a real world context using an ontology? Let us assume that we wish to represent a component of a building such as a conference room in the computer. Until recently, in a *data-centric* software environment, we would have treated the conference room as a three-dimensional geometric entity that can be described in terms of points (i.e., x-y-z coordinates), lines, or surfaces. While this may be satisfactory for displaying different internal views of the building space and even generating animated walk-through sequences, it does not provide a basis for the computer to reason about any aspect of the space, such as that a conference room must have a door for it to be usable. To provide the computer with such a reasoning capability the particular entity, in this case the conference room, must be represented in the computer as an information structure that describes conference room as an integral component of a building. This can be achieved by storing in the computer the word building and associating this word with some characteristics such as: a building is a physical object; it is made of material; it has height, width and length; consists of one or more floors; has spaces on floors; and so on. Then further defining spaces with characteristics such as: enclosed by walls, floor and ceiling; with walls having at least one opening referred to as a door; and so on.

In such an *information-centric* software environment the same conference room would be stored in the computer as part of the building ontology, allowing the following more intelligent user-computer collaboration:

Computer user: I would like to represent a component of a *building*.

Computer software: Loads its stored *building* ontology into memory.
Asks user "What kind of a *building* component?"

Computer user: A *space* of type *conference room*.

Computer software: For how many persons?

Computer user: Up to 16 persons.

Computer software: Suggested space size is: 16 ft (length), 14 ft (width), 8 ft (height).
Suggested furniture: 6 ft by 3 ft table, 16 chairs, screen, white board.
Other features: There must be at least one door.

As can be seen from this user-computer interaction, the computer software by virtue of its internal information structure has some understanding of the meaning of a *building* within the context of its characteristics and the relationships of its components (i.e., floors, spaces, walls, openings, and furniture). This endows the computer software with the ability to collaborate and assist the user by reasoning about the relationships between the data entered by the user and the context contained in the relatively simple information representation provided by the *building* ontology.

6. Connecting the Dots

The potential impact of this kind of computer-based reasoning capability on current data-based intelligence collection and analysis practices is profound, to say the least. For example, let us assume that during a typical four-week period the following four messages (Figure 5) have appeared among the hundreds of thousands of data items that were collected by the various intelligence agencies: (message A) *terrorist warning from Egyptian counterintelligence agency*; (message B) *suspected Middle-East terrorist apprehended on entry into US*; (message C) *meeting between suspected terrorist operatives at O'Hare Airport in Chicago*; and, (message D) *explosives theft in Chicago*. Although the messages come from different sources and are collected by different intelligence agencies, they are automatically time-stamped and fed into a message network (Figure 6) that is presumably accessible to all such agencies.

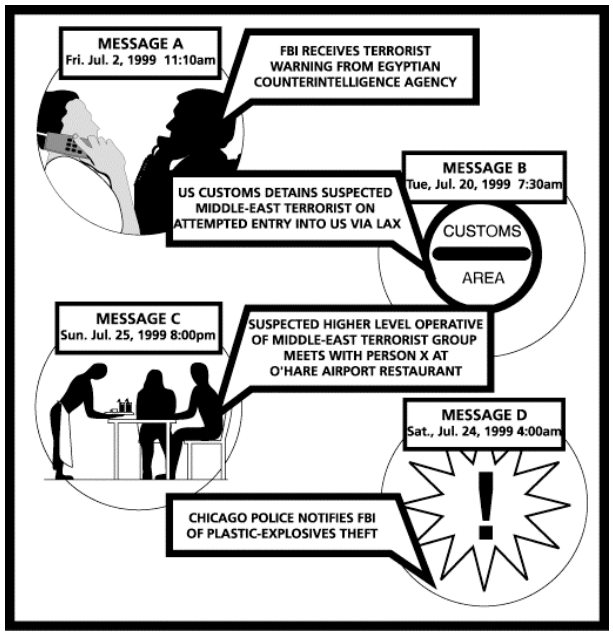


Figure 5: Four typical intelligence messages (***data-centric*** software environment)

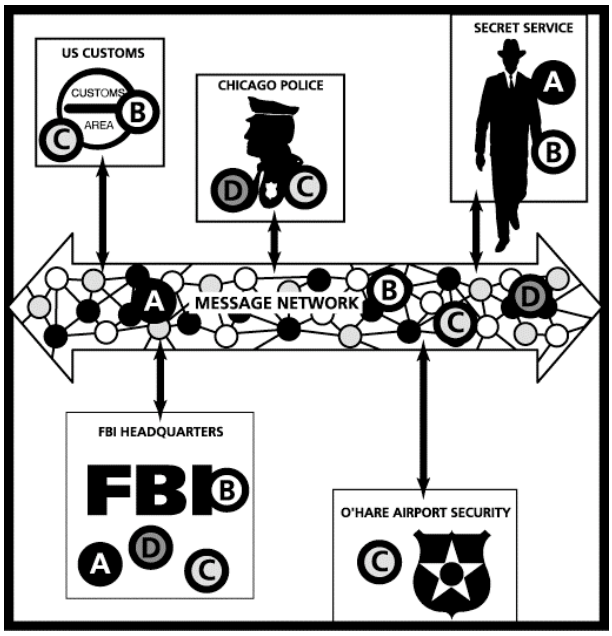


Figure 6: Manual processing of messages (***data-centric*** software environment)

There are several serious problems with this method of collecting and subsequently analyzing intelligence data in the current *data-centric* software environment. First, virtually all of these raw data items have to be analyzed and evaluated by human operators. The absence of relationships (to provide context) prevents any really useful automatic filtering to be implemented. Only limited data-processing methods such as keyword searches and indexing, can be employed. This

places an enormous burden on limited human resources. As a result valuable human intelligence personnel are literally burned out at the lowest level of intelligence analysis, leading to a tendency for the data collection activity to be restricted.

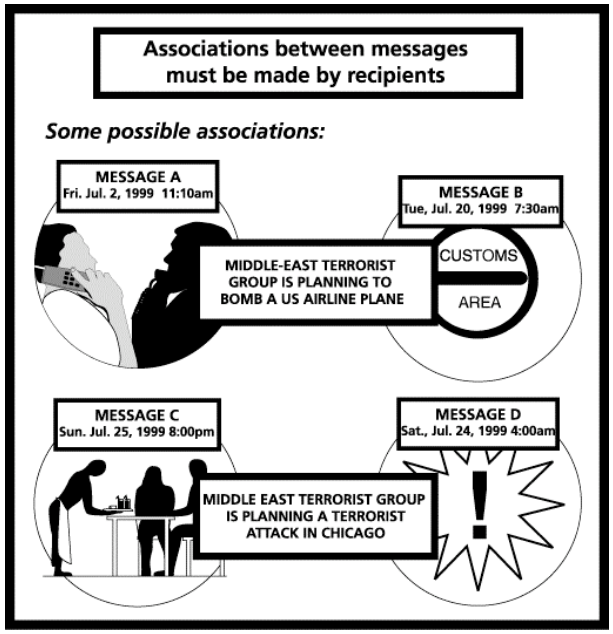


Figure 7: Manual processing of associations (***data-centric*** software environment)

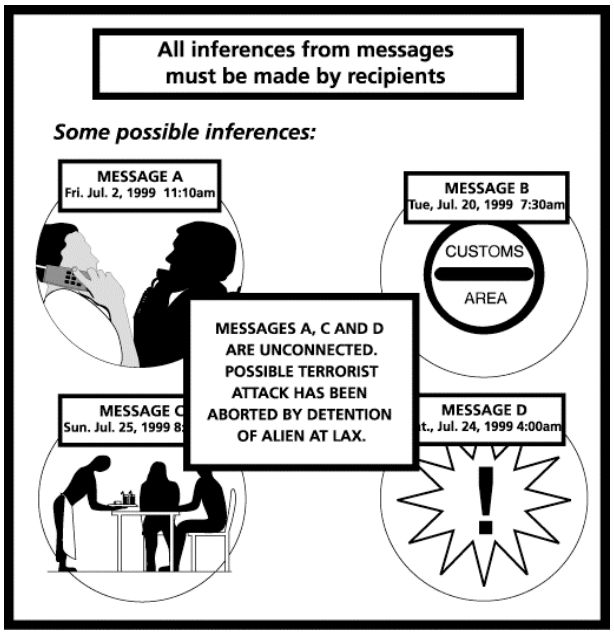


Figure 8: Manual processing of inferences (***data-centric*** software environment)

Second, there is no guarantee that all agencies will receive all four of the data items. Since any associations between these and other messages have to be made by human analysts (Figures 7 and 8) there is a good chance that some of the possible associations will be either overlooked, or not pursued for lack of available manpower. Yet these associations are critical for the transformation of data into information and the detection of patterns through appropriate inferences. Even if the available human resources allow first level associations to be established, it is unlikely that many second and higher level associations will be developed due to human labor limitations and time constraints.

Nowhere are the shortcomings of a data-centric software environment, in which the computer-based systems have no understanding of what is being processed and virtually all interpretation tasks must be performed by human operators, more apparent than in the intelligence community. While the *information-centric* building blocks that would allow computer software to play a far more powerful role in intelligence analysis and evaluation processes have been available for at least two decades, the intelligence community like most other computer users has been reluctant to take advantage of these potential capabilities. Clearly, this reluctance is not based on technical obstacles but on the innately human resistance to change. Unfortunately, only a compelling reason such as the post-September 11 (2001) terrorist threat will provide the necessary incentive for the timely adoption of *information-centric* software design and implementation principles.

The benefits of such a paradigm shift will be immediate with startling results, even in the initial relatively primitive implementations. Returning to the previous example, the four typical intelligence messages would be processed very differently by ontology-based software that provides sufficient context for some degree of automatic reasoning by software agents. What is

immediately obvious in Figure 9 is that key elements of the messages are treated as objects with attendant characteristics and relationships.

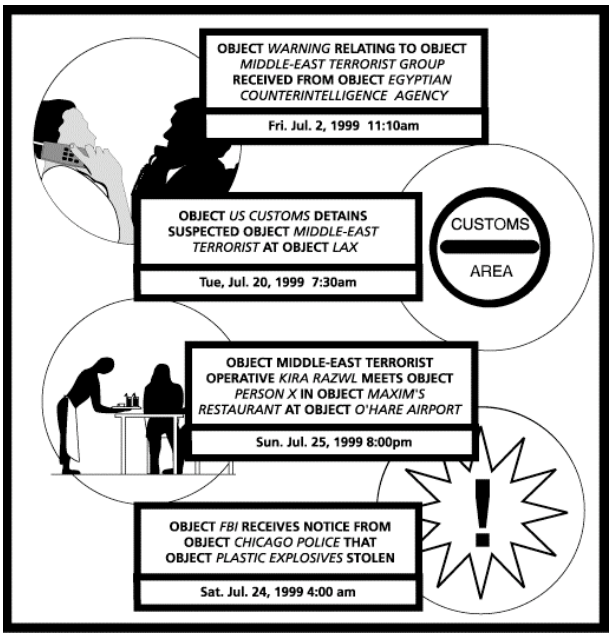


Figure 9: Four typical intelligence messages (***information-centric*** software environment)

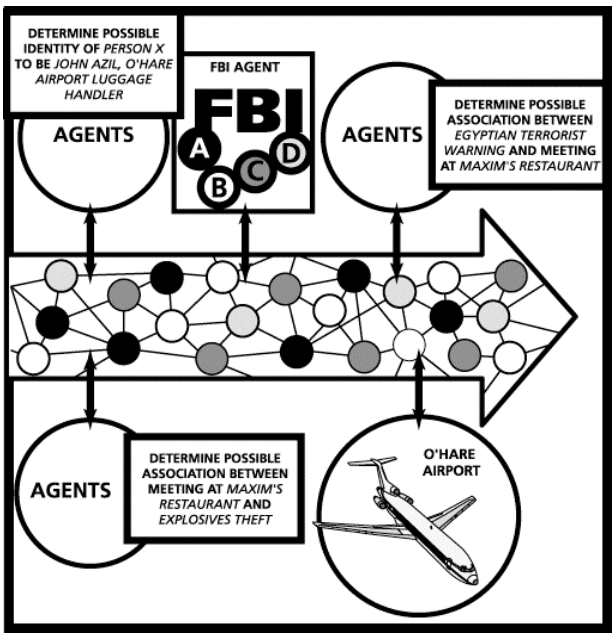


Figure 10: Automatic processing of messages (***information-centric*** software environment)

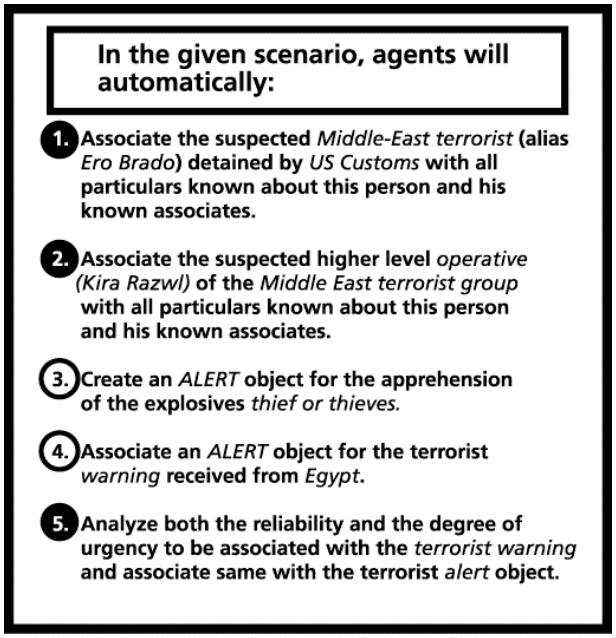


Figure 11: Automatic processing of associations (***information-centric*** software environment)

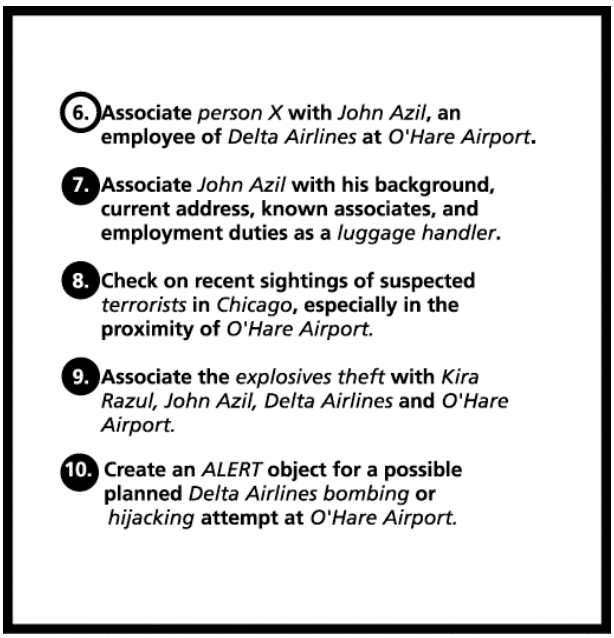


Figure 12: Automatic processing of inferences (***information-centric*** software environment)

For example in the first message, by being represented as objects within an information model the Egyptian Counterintelligence Agency, the Middle-East Terrorist Group, and the Warning itself, provide a rich context from which inferences can be drawn. This context includes not only

the information that is already available about the Middle-East Terrorist Group and the Egyptian Counterintelligence Agency, but also the relationships among these entities and other entities (i.e., objects) represented in the information framework. This allows software agents (Figure 10) to automatically explore numerous associations and possible conclusions in parallel and without initial human assistance. As a direct consequence scarce human resources can be employed at higher levels of intelligence analysis and evaluation, after most of the tedious low level filtering and data correlation tasks have been accomplished.

A typical sample of the kinds of automatic analysis and reasoning that could be performed by software agents in the given example is shown in Figures 11 and 12. Not only would the agents be able to automatically access relevant databases to enrich the information environment, but they would also be able to correlate data sources, form associations, detect patterns, and explore possible conclusions. The number of software agents performing these tasks in parallel would vary depending on current needs and hardware capabilities. In other words, agents could be cloned by other agents, or even themselves, as demanded by the workload.

7. Concluding Remarks

Endowing computer software with a virtual representation of the real world context that is required for the automated interpretation of data will not only allow us to apply far superior security and information assurance methodologies, but also provide the basis for a major leap in the exploitation of computer-based capabilities.

Moving from data-processing to *information-centric* software constitutes a paradigm shift in capabilities and human attitudes. It is not a question of whether this paradigm shift will take place, but how long will it take. While the miniaturization of electronic components has advanced in a proactive mode at an astounding rate over the past two decades, the exploitation of these newfound hardware capabilities with commensurate advances in software has progressed more slowly in a reactive fashion. The principal reason for this reactive inertia is the natural human resistance to change. We typically change only in response to a serious threat or to take advantage of an obvious and greatly enticing opportunity. Both are present at this time. We are facing a compelling and dangerous cyber threat and we are beckoned by an opportunity of great economic gain.

References

Biermann A. and J. Feldman (1972); 'A Survey of Results in Grammatical Inference'; Academic Press, New York.

CBS (2009); 'Cyber War: Sabotaging the System'; CBS 60 Minutes Series, 8 November.

CNN (2009); 'Hackers stole data on Pentagon's newest fighter jet'; Mike Mount, CNN.com/US, 21 April.

Cohen B. L. and C. A. Sammut (1978); 'Pattern Recognition and Learning With a Structural Description Language'; Proceedings Fourth International Joint Conference on Artificial Intelligence, IJCPR, Kyoto, Japan (pp.394).

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

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.

Winston P. (1970); 'Learning Structural Descriptions from Examples'; Technical report AI-TR-231, MIT, Cambridge, Massachusetts, September.

Zetter K. (2010); 'Hackers Targeted Oil Companies for Oil-Location Data'; 26 January (www.wired.com/threatlevel/2010/01/hack-for-oil).

Knowledge-Based Collaborative Architectural Design: Abstractions, Filters and Process Improvements

Gianfranco Carrara[1], **Antonio Fioravanti**[1], **Umberto Nanni**[2]

[1] Dept. Architettura e Urbanistica per l'Ingegneria
Sapienza University of Rome
Via Eudossiana, 18 – 00184 Rome – Italy
Tel. +39 0644585165; Fax +39 0644585186
gianfranco.carrara@uniroma1.it; antonio.fioravanti@uniroma1.it

[2] Dept. Informatica e Sistemistica
Sapienza University of Rome
Via Ariosto, 25 – 00185 Rome - Italy
Tel. +39 0677274020; Fax +39 0677274129
nanni@dis.uniroma1.it

Abstract

The increase in the complexity of the building process is leading to a parallel general reduction in product quality, commonly ascribed to the inadequacy of the routine design methods and tools. Indeed the latter make their overall integration more difficult and impose serious constraints on design creativity, while they do not help design considered as 'the ability to choose from different solutions'. It is generally recognized that the solution of the problem lies in efficient forms of collaboration among all the actors involved in a project. However none of the forms and tools proposed hitherto has been found satisfactory.

This paper shows how the essential basis of all forms of collaboration lies in the representation and management of the knowledge activated along the design process. Then an innovative distributed Knowledge-based system *aiming at an effective and creative collaboration among the actors. By virtue of the interoperability established among the various semantic universes it enhances the level and the quality of the information exchanged among the actors while managing not to change their operating modes. We show how the proposed organization of the Knowledge Structures provide a basis for improving the design process.*

Keywords: *Architectural Design, Collaboration, ICT, Knowledge Based Systems, Ontologies.*

1. Limits of Current Building Design Process

The interdisciplinary and dynamic nature of Architectural Design and Building in general clearly reveals the limits of conventional design in coping with the rapid changes taking place in this context (in the broad sense) in which the number of professional profiles is larger than any other industry and, between construction and the actual running, absorbs about half the global energy consumption.

The increasing complexity of the building process and product makes them ever more difficult to manage and, at the same time, there is an almost imperceptible but constant reduction in the final quality of buildings during the process.

The design involves a diverse set of actors. In the present paper all the physical and virtual subjects that in any way take part in the design process, i.e., humans, firms, designers, users, clients, intelligent software agents and assistants, are hereafter denoted as 'actors'. Process/product complexity seems to be a direct consequence of our technological culture, bound up as it is with continuous growth and segmentation of technical and procedural rules, codes, changes in the cultural and environmental contexts, fragmentation of activities into parallel and sequential phases, increasing size of building operations, higher performance levels required for the whole and the separate parts of the product and thus becomes an unavoidable component of the daily work of professionals. The resulting quality of the building obtained through this type of process is too often unsatisfactory in terms of the formal results, of the failure to achieve the technical and functional objectives, of excessive energy consumption, of unsustainable environmental impact, and of cost and time overruns.

These two critical factors are linked and interdependent. The inferior quality of the building product is generally a direct result of inadequate design, often the result of problems of communication and understanding among the various actors in the project and is part and parcel of the process itself.

Generally speaking the concept of *design* increasingly extends into a large multitude of sectors as it is necessary to try and foresee the often unpredictable changes resulting from new inventions and changes in technology, tools, methods and social customs: *design is -pervasive vis-à-vis all the problems.* The present study therefore takes into account the general features of the *essence of design.*

The tendency to speed up the design process by holding other design hypotheses in low consideration often leads to raising and spreading conflicts among actors. Moreover an impoverishment of the final quality of the project is observed vis-à-vis the initial idea of specialist actors. This is the result of both the adoption of simplified, conventional or relatively non innovative design solutions that do not lend themselves to providing higher performance and the practice whereby different actors' design solutions tend to be simplified in order to avoid reciprocal misunderstandings related to innovative design solution.

The problems outlined therefore demand to seek other forms of process which allow project efficiency to be improved by cutting down the time required and by favouring the development of creative ideas.

In the quest for a solution of the aforesaid problems lies the motivation of the research presented here aimed at conceiving new means for facilitating an effective interaction among actors in the current complex cross-disciplinary building process.

Complex building design is nowadays a process characterized by a high degree of interdisciplinary activities (Bjork, 1999; Bjork, 1992; Carrara et al., 2004; Carrara & Fioravanti, 2002; Carrara & Kalay, 1994). This kind of process requires a high degree of collaboration. Each actor has its own language, concepts and skills. Information technologies have provided since the early adoption a valuable support for their individual activity. This support has led to the creation of ad hoc tools for CAD, structural design, design systems, which have been progressively

refined over several generations of technology as tools for the single specialist. In contrast, the interaction required by these actors in the design process for exchanging requirements results, and constraints arising from these, for many years has not evolved significantly, leaving the burden of interoperability to the traditional practice-based design, and resulting in either a superficial knowledge or a costly inspection of the work done by other actors. On the other side, collaboration in building design is an inherent necessity in the AEC sector, as any building is a singular, integrated and complex system in a given context with interleaved problems. Collaboration allows the various solutions by the different actors to converge towards a single overall solution, encouraging creativity through interactions among the different skills (Gross et al., 1998; Kvan, 2000; Jeng & Eastman, 1998, Kolarevic et al., 2000).

However collaboration is hard to apply in highly complex projects and processes. Moreover, the difficulties facing collaboration increase in large-scale projects as a result of the scattering of actors in space and time, the different languages involved and, above all, due to the *symmetry of ignorance* whenever the shared knowledge proves to be insufficient.

Efficient collaboration among actors in the design process means that all actors must be able to propose the solution to the specific problem they are responsible for solving, in such a way that the other actors can understand it in order to be able to modify their own solutions to adjust to the received suggestions or to consciously object and eventually reject it. For this to be possible several fundamental criteria must be satisfied: communication and acquisition of information must be correct, efficient, secure and unambiguous; the presence of as much *specialist knowledge* as is required by the complexity of the planned product; the presence of *knowledge shared* by all the actors involved to allow them correctly to interpret the information exchanged and to understand each other; a semantically and technically correct link between shared knowledge and the universe of specialist knowledge of the various actors.

All this indicates that the fundamental bases of collaboration lie in *knowledge*, and in the *way* it is exchanged among actors, regardless of the tools used in the design process.

2. Limits of Current IT-Tools for Architectural Design

The introduction of ICT science has radically modified the way information is transmitted in the design process: drawings and documents are transformed into data structures that imply the use of new tools to produce and transfer them.

A wide variety of computing and representation software is available on the market, which is capable of performing even relatively complex tasks within well-defined disciplinary boundaries although designed to enhance the capacity to verify a given design approach rather than to help find a design conception. These software applications are actually of no help in design collaboration, and indeed make it more difficult. The lack of mutual understanding is mainly due to the low semantic level afforded by the application programs used by the actors and by the inadequate degree of interoperability of the software used. Moreover, different actors provide different interpretations of the same object; these situations are a cause of misunderstandings that are all the more detrimental the greater the degree to which the actors continue to develop their own specific design solution which often turns out to be incongruent with that of other actors.

Such difficulties are due both to the *lack of an overall model of the building* and *of the design process* that is representative of their complexity and to an *inadequate formalization of information* pertaining to any individual actor and exchanged among the various actors.

Indeed an effective formalization of information exchanged along a design process remains an unsolved problem. Each actor makes use of low level formalized semantic information and operates on it by deploying his/her/its own professional skills and experience.

In the AEC community several efforts have been devoted to overcoming these difficulties in order to integrate competencies in a single application program and to share knowledge. Among the various initiatives, we mention *BIM* and *IFC*.

BIMs (Building Information Modeling) are product models recently driven by several CAD system firms mainly (Autodesk, GraphiSoft, Bentley, Nemetschek, etc.) which can describe the form (e.g., geometric information and its relationships) and attributes (e.g., physical characteristics) of a building throughout its life cycle. BIMs define a building with proprietary formats conceived from a top-down point of view, focused just on components and not on the process or on the building as a system, that are a source of intelligent information about a building.

To achieve a better interoperability of industry-wide application software, an efficacious basis for information sharing between different BIMs, has become necessary and urgent. To this end a second initiative has been developed following a different approach: Industry Foundation Classes (IFC). It is an open XML standard (OOP) conceived bottom-up with non-proprietary data model specifications, proposed by International Alliance for Interoperability (AIA), that is emerging very slowly among involved industries. IFC aims at granting software interoperability while exchanging more significant project data, so that nowadays CAD applications by major software houses can (with some difficulties) import and export their proprietary formatted files from/to IFC files. Such specifications represent a data structure supporting a digital project model useful in sharing structured labelled (more comprehensible) data across applications, but they are neither intended for *design needs* nor for *mutual understanding* among actors, but mainly for production needs.

So far, exchanging contents among commercial applications has been very difficult. Indeed the export of proprietary BIMs, from their own file formats to the corresponding IFC one, is not equivalent due to their own different primary conceptual models of the building. Moreover, even though different specialist actors use the same integrated application tool (e.g. Revit, Triforma, etc.), the entities they consider can have different meanings as they belong to different specialist domains. For instance a window assumes different meanings and representations when related to different specialist domains (such as an architect's, structural engineer's, building scientist's and so forth) as the former are closely linked to underlying models of the aspects of reality considered.

BIMs have an important role in creating and coordinating components as parts of a building, but actually do not provide any concept of the building as a 'system' (structured set of components with functions aimed at a goal) such as architects or engineers have had for centuries. Moreover other difficulties arise from the fact that BIM data must co-exist with a number of programs with different task-oriented models, all essential in defining detailed, but only partial information of a project.

IFC is based on a central model that can be either partially or entirely shared by participants, but must be accepted as a whole, as being totally coherent (it is not scalable from this point of view). Although its approach supports different visualizations of the same '*component*', it is focused on converting and updating 'components' from multiple sources, at the level of the applications, into a generalized description of the entire building.

Current interoperability design problems related to commercial application programs are actually solved within the domain they were built for as very often they all have a similar, but specific, point of view: that of the person who first modelled the phenomenon, probably some thirty years ago. In conclusion, the model underlying the application programs of a specialist domain allows *data exchange* but not the *inferring of concepts* from the application programs themselves, as these concepts *are implicit and tacit between the actors of that domain*. Such a problem is not a big deal between actors of the same discipline-specific domain, but it is crucial to solve it in cross-disciplinary design in order to allow and improve collaboration. Its solution is not concerned with mere interoperability formats: it is above all concerned with how to makes concepts related to products explicit and understood by all the actors. How to overcome the *symmetry of ignorance* is still an open problem.

At present an export of application programs to/from the common low ontology level (IFC) for machine interoperability purposes only, is becoming available - to the extent that software houses support new IFC specifications. The dominant way of using IFC specifications (low-ontology level) today is still a one-direction batch translation of large data sets from an application into the common language (IFC) and vice versa. Collaboration using IFC specifications exists in the industrial practice, but is based on 'ad-hoc' procedures agreed between single specialists for a single project. Indeed the IFC model servers so far implemented provide limited collaboration support and the existing model servers do not support adequate management of the instance versions (with different meanings) of various specialists.

3. An Innovative Approach to Building Design

In order to overcome the current limitations of collaborative design, we propose improvements to the organization and representation of knowledge and the consequent information exchange among actors together with a cycle for quality improvements both for tools and processes in the practice of collaborative design.

In order to develop new tools that can efficiently support actor's design work it is necessary to reflect on *what* is required along the design process in order to: allow actors to retrieve that part of their own knowledge considered relevant for the project, make all the actors understand each other correctly so as to set up an effective collaboration in order to activate all this in a complex process.

Knowledge used in the design activity can be partitioned according several criteria; we will consider the following:

Common vs. *Specialist*: this has to do with the nature of the information. Each actor has a specialist knowledge; on the other side, the common knowledge has to be fully understandable to all actors;

Project Independent vs. *Project Dependent*: the project independent knowledge concerns notions that are not contextual to the project at hand (such as concepts, rules, constraints,

patterns...); the project dependent knowledge is accumulated since the early stages and increased throughout all the phases of the project; at the end of the activity the project dependent knowledge may be imported, fully or in part, in the project independent knowledge base;

Private vs. *Shared*: when the actors share some portion of knowledge they are better able to interact and understand each other. Thus to increase the efficiency of the design process, each actor has to share with all the others a part of his/her knowledge deployed in the project that is of common interest; sharing an information is a deliberate act, depending on the will (or duty) of the single actor;

Abstraction: a hierarchy of concepts allow to capture the finest details of the design (namely, the data values that characterize each single property of any entity in the project), and their aggregations, up to high level entities (e.g., those useful to express properties and performances in the early requirements).

The proposed structure is based on a formal representation of knowledge by means of *ontologies*, encompassing both the formal structure of the entities (i.e., meanings, geometry, properties, relations, etc.) and the formal models that allow simulations, verifications and reasoning to be defined and processed. A *Knowledge Structure* (KS) is composed of a set of Entities, each of which is related to an *Ontology* (its definition) and has a S*emantics* (its meaning). Each entity can have a set of *Properties* (geometric, physical, values) and *Attributes* (function, methods or computing programs), a set of *Belonging Relationships* with other entities (part-of / whole-of), a set of *Inheritance Relationships* (class-of / is-a), a '*situation*' (or '*Condicio*', Carrara & Fioravanti, 2004, pg. 430) dependent set of *Rules of Compatibility* with other entities (check-list, adjacency-list, etc.), *Inference Engines* (IEs) to activate and manage constraints, all of which are formalized into a syntactically coherent IT structure. The aforementioned KS is actually a 'system' if the structured entities present in a KS aim at a goal: e.g. habitability, energy saving, constructability, etc. A goal is achieved through several objectives and sub- objectives. E.g. habitability includes the usability of the spaces, the ergonomics, the space brightness, reciprocal disposition of spaces, space relationships with the outside, etc.

To make it possible entities in a domain are related each other by means of specific relationships and IE the *Relation Structures* (RS). In such an environment any actor involved in a design process manages his/her/its own entities in order to attain his/her/its own specific goal. The key element of a KS, anyhow formalized and structured in the fields of architecture, energy saving, sustainability, buildings stability, etc., is based on the *definition* of entities involved in a domain, therefore in its ontology.

For all these reasons the scientific community had a new interest in the field of *ontology* that provides a valuable support for representing and sharing terminology, concepts and relationships within a given domain, so that an increasing number of communities (Ugwu et al., 2005) of experts develops ontology as an underlying base for their work, including collaboration in design. Actually in the growing area of network services new approaches to composition and orchestration of services are based on ontologies for representing their definitions, i.e. for disambiguating queries.

At present most entities of a specialist ontology are typically not explicit and are inherent in the model of the phenomena they refer to (Dakros and Knox, 2004), so that in commercial tools part

of the knowledge is implicit, hidden-coded in application programs and is neither openly available nor fully understandable to the actors, so that an *Explicit semantics* ('situation' dependent) is needed to make entities understandable by humans and tractable by computers. As actors take into consideration project entities at different levels of abstraction, from the data level to the reasoning level, an *ontologies based methodology* allows the actors coherently to use different levels of abstraction and/or to exploit a conceptual interoperability.

All Knowledge required in a design process can be split into *Common Knowledge*, which allows all the actors to communicate and essentially to understand each other, and as many sets of *Specialist Knowledge* as there are disciplinary domains involved in the specific design process, depending on the type and stage of the project considered. *Specialist Knowledge* therefore includes a set of entities in a specific disciplinary domain (some of them can be the same entities of the Common Knowledge) with specific semantics, properties, attributes and relationships.

The specific tasks of S*pecialist Knowledge* are: to perform simulations and behaviour verifications related to its own goal of entities, to infer reasonings from entities by means of its own RS and IE, to notify suggestions and notifications to the specialist actor and to the other actors concerned.

4. Ontologies as a Basis for Knowledge Structures

An ontology is a representation designed to support humans and computers in order to represent knowledge with agreed meaning. Basically it contains a set of words, relationships, meanings, in a format that must be readable (for human use) and formally defined (to be used by computers). The ontologies considered in Knowledge Structures include all and only the entities retrievable in any possible object that is definable in the design process. Referring to building design, we will consider two fundamental structured ontologies: that of *the spaces and their aggregations,* which in a project make up the so-called 'spatial system', and that of *the physical elements (components) and their aggregations* which in a project make up *the constructive apparatus*, defined by UNI (Italian Standards Organization) as the '*technological system*'. These two systems as a whole make up the *ontology universe of Project-Independent Knowledge*.

In order to guarantee collaboration and the proper integration of the specialist design solutions into the overall one it is crucial to set up correspondences between the ontology of the entities included in the *Common Knowledge* and those in any *Specialist Knowledge*. Specialist Knowledge entities often have so many more characteristics and properties than the corresponding ones of Common Knowledge as to be defined as 'heavy'. Specialist Knowledge, in addition to the (light) *ontologies* corresponding to the ones included to Common Knowledge*,* also contains *specialist ontologies*, which are specific to the specialist domain and are not present in the Common Knowledge, although they may be present in some other Specialist Knowledge. Private ontologies can refer to aggregations of components that are meaningful only for the specific domain, e.g. an office suite, which is meaningful for the architect but is just a collection of rooms for other specialists.

Through this identity correspondence the entities (or their aggregations, as well as their characteristics or properties) which are specific to a disciplinary domain and formalized in *Specialist Knowledge Structure*, are related to the corresponding entities in the *Common Knowledge Structure*, which contains all and only what is required to be known by all the actors.

Inside a *specific project* – a design solution – the ontology set of *Spaces* and the one of *Components* are closely interrelated and established by a set of reciprocal relationships where the characteristics of one are determined by and in turn define the characteristics of the other. Any '*situation'* (or '*Condicio*', Carrara & Fioravanti, 2004, pg. 430) of a design process determines a sub-set of the all possible solutions as it depends from values of properties of entities (classes), defined by the actors, the site, the design phase. Any design solution (an individual), called *'instance'*, whether *specialist* or *overall,* is made up of *data* (specific values) and therefore is clearly distinct from the Knowledge (a structured set of classes) used to build it, which is made up of *metadata* (classes).

The *overall design solution* (the overall instance), which does not necessarily demand to be congruent until the end of the process, is made up of the *specialist design solutions* of all actors (the personal instances) and of the *Common Knowledge solution (Common instance)*.

As one of the assumptions of this work is that actors cannot overcome the symmetry of ignorance barrier at a data level or at a low semantic level by means of usual application programs, such a goal can only be achieved by means of:

- mapping concepts of ontologies of different domains by means of a previous agreement among actors and/or by means of rules to state same entities (Cheng, 2008);
- managing concepts of ontologies of different abstraction levels by means of inferential engines and intelligent agents in order to have an effective support in design.

For the first point, in order to allow mapping among the entities belonging to different ontologies (ontology mapping), each specialist should be provided with his/her/its own ontology while all these should be partially overlapping. The resulting intersection set, defined as *Common Ontology* (fig. 1), provides the base through which actors can understand each other. Ontologies are dynamic and incremental as they allow actors to capitalize knowledge and expertise.

For the second point, in order to relate current problems to past experiences knowledge has to be stored in an appropriate format at the correct level of abstraction.

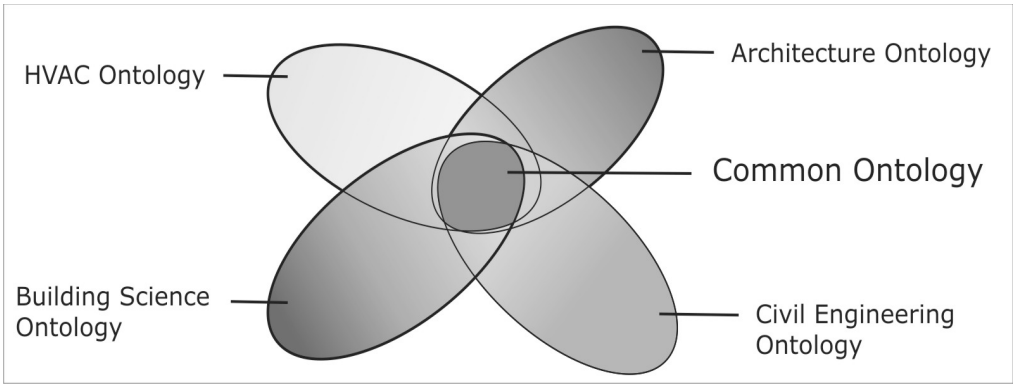


Figure 1 – Common Ontology as the intersection set of several Specialist Ontologies.

Here we see a *Structure of Knowledge* used along a design process, based on *layered levels of intelligence* (fig. 2): an IFC-based *Lower-Ontology-Level* (LOL), a rule-based *Upper-Ontology-Level* (UOL) and a logic-based *Deductive-Reasoning-Level* (DRL) (Nguyen et al., 2005). UOL allows its own logic/ algorithmic rules - that can adapt themselves to their '*situations*' (or

'C*ondicio*') – to explain parametric objects, constraints, etc. DRL, with its deductive capabilities, applies inference rules and intelligent agents to UOL entities. This process is facilitated by using IFC standards and is triggered whenever ontologies in an occurring 'situation' are instanced and LOL ontologies of different actors are related to each other at DRL. By means of such a mechanism constraint rules can be transitively chained as much as possible.

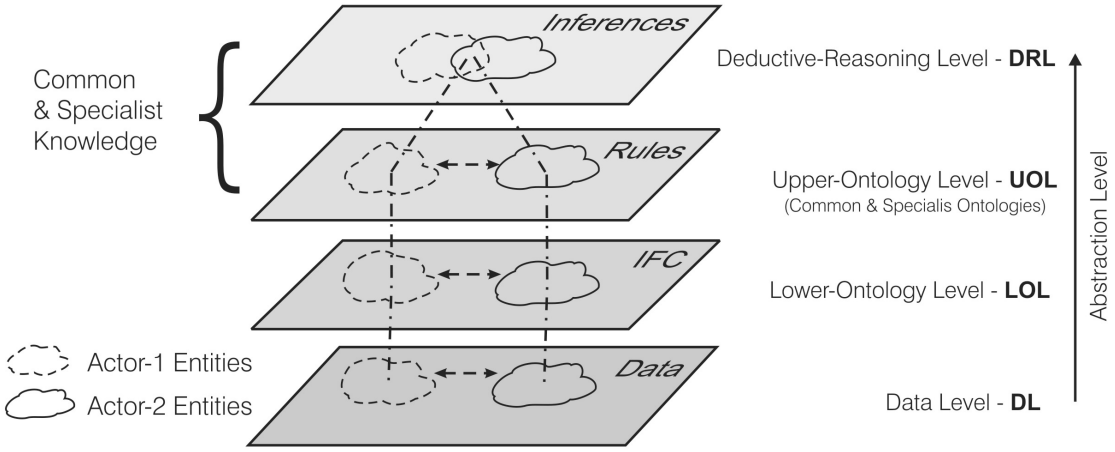


Figure 2 – Structure of design Knowledge: Ontologies at Deductive-Reasoning Level map different Actor's ontologies at the Upper Ontology Level so that an inference mechanisms can be applied to rules.

With respect to the use of inferential engines and intelligent agents, the approach described here differs from the exhaustive and integral approach hitherto developed – consisting in importing and exporting all the information in an agreed and identical format such as IFC (which in any case can be a basic reference for semantic matching) – as it is based on:

a) exporting strictly necessary information among actors by means of the Common Ontology;

b) leaving to an Upper-Ontology level based tool the task of linking representations of the same entities made by an actor at different levels of abstraction (vertical interfaces);

c) using entities at the Deductive-Reasoning level to map entities of different actors of the Upper-Ontology level (horizontal interfaces).

a) In contrast to a centralized database model, a distributed ontologies model has been developed based on a *Common Ontology Domain* and several *Specialist Ontology Domains* (Fioravanti & Carrara, 2007; Carrara et al., 2004; Carrara & Fioravanti, 2001; Leeuwen & van der Zee, 2005). Each specialist actor's domain retains its own ontologies in the most appropriate form for actor needs and expertise, while an appropriate interface translates its own ontologies into/from the Common Ontology Domain.

b) The research work has defined an ontology based model that supports the actor's design by linking his/her/its heterogeneous abstraction levels – whose formalization is oriented towards different tasks – with the data of his/her/its usual application programs, so that the model can detect any data inconsistency (DL), incoherency of constraints (LOL), incongruence of goals (UOL) within any actor's domain.

c) Ahead of time actors can acknowledge the implications of their proposed project solutions, considering other actors' points of view, constraints and goals by means of inference

mechanisms at the Deductive-Reasoning Level. The latter are able to establish a mapping among the same entities present in different domains so that constraints (e.g. a constraint on a dining room surface), rules, goals among ontologies belonging to different domains can be chained.

Incoherencies are detected by an inferential mechanism contained in the DRL as it points to (if one exists) a common entity (e.g. a pillar) in the Common Ontology by mapping the ontologies (and their structures) of different domains, then checking contradictions among the latter (e.g. different acceptability ranges of dimensions); subsequently it reports feedback information to the actors involved (who have pillar constraints in their own Specialist Ontology) so that they can provide the necessary action.

The dynamic and semantically-specific representation detecting incoherent/favourable situations by means of a constraint rule mechanism can allow them to be highlighted and managed in real time. At the same time it allows actors to make alternatives, more consciously reflecting on the consequences of their intents. In this way knowledge spreading throughout the networked ontology-based environment makes actors more aware of other specialist constraints, allowing them to make more participative and shared choices.

The integration of the specialist actors' design solutions (instances) translated into sub-sets of the overall design solution (instance) can give rise to inconsistencies and conflicts among instances belonging to different workspaces or to ontologies of different domains.

5. Knowledge 'Translation' and Filters

To interface common knowledge with specialist knowledge a specially conceived mechanism is required. This is made up of a *Filter* that allows *any specialist actor's design solution* (all information that is specific to each *actor)* to be directly and automatically 'simplified' and 'translated' into the '*common language*' of the overall solution. Since the latter is correctly read and interpreted through the Filter, it leads to mutual understanding among all the actors involved in the design process. The representation of this 'simplified translation' is called '*Common View*'.

Common Knowledge, including all and only knowledge shared and agreed in times by *all* the actors, takes on a fundamental role in the current complexity of the design process, too often lacking in practice, allowing the actors to interact and to mutually understand each other at a basic level, with respect to *the design solution* as it is progressively processed.

The integration of the *personal instances* into the *overall instance* is achieved through the Filter mechanism that acts at the level of classes (knowledge filter) and at the level of *instances* (data filter).

The *Knowledge Filter* works at the level of concepts (ontologies, properties, relations, values) and acts as an intermediary between each Specialist Knowledge Structure and the Common Knowledge Structure: it recognizes among the entities selected by each actor in his/her/its own Specialist Knowledge Structure those that are present in the Common Knowledge Structure, selects them and determines the corresponding sub-set of them inside the latter.

The second filter, the *Data Filter*, works *at the level of individual data* and acts as an intermediary between each individual data structure representing a *personal instance* and the data structure representing the *overall instance*: it is triggered by the first filter and recognizes among the data of each *specialist instance* those corresponding to the entities selected by the

Knowledge Filter in a given 'Condicio' and translates them into a sub-set of the data structure representing the *overall instance*.

In order to ensure a dynamic and interactive knowledge exchange among the actors, which is an essential prerequisite for an effective collaboration, a suitable organization of the *Design Workspace* is required. This term represents the '*place*' in which the design activity is performed and metaphorically corresponds to the 'professional office' in which actors work in the conventional design process.

The distribution and centralization of Knowledge corresponds to a similar arrangement of the *Design Workspace* in which the actors are called upon to work. This is therefore divided up into a number of *Private Design Workspaces* - specific to each individual actor and corresponding to the number of specialist actors involved - as well as into the *Overall Design Workspace*.

Design Workspace presents a distributed structure of *Private Design Workspaces*, each one referring to one of the numerous actors involved in the design process for the implementation of specialist solutions. This structure is directly linked to a *Overall Design Workplace*, shared by all the actors, so that they can visualize the merging of all the simplified partial solutions and recognise which one can be subjected to verification.

During the process each actor can therefore create or modify his/her/its own *Personal Design Instance* in his/her/its own *Personal Workspace*, using his/her/its own specific Specialist Knowledge and hie/her/its own personal tools. This instance is part of the *overall design instance* constituted by the merging of all the *partial specialist instances* in the *Overall Workspace*.

During the whole design process each actor is able to verify his/her/its *own personal design instance* using their own *Specialist Knowledge* (his own ontologies and deductive capabilities) and whenever it is deemed satisfactory he/she/its can, by means of the filters translate it into the '*common language*', combine it with the *personal design instances* of other actors and verify it in the Overall Design Workspace through *Common Knowledge.* This is still a personal 'test phase' of the *personal design instance* with respect to other actors' constraints as far as actors do not consider to show the other actors her/his solution with related advice or warnings

When actors deem their own partial design instance satisfactory (whether verifying or not constraints), s/he can '*publish*' her/his own instance in the *Overall Design Workspace* so that it becomes visible to all the actors and can be queried.

The *published* design instance is simply one version of the project's evolution, which does not have to be totally consistent. It is possible, at every stage of the process other than the final one, for the design instance (the project) to be inconsistent both internally and/or with the other instances. The process finishes when actors agree that a design solution is acceptable.

Through the interaction among the personal instances worked out in the *Private Design Workspaces* and shown to all the actors through the *Overall Design Workspace,* a *flexible and continuous interaction* is established among the actors in spite of their reciprocal interdependence, thus *setting the conditions for a genuine and efficient collaboration*.

Summarizing, the described Knowledge Structure is operationally subdivided into two basic levels – that of *knowledge* and that of *data*.

The upper level (*Knowledge level*) can be conceived of as split into a *Project Independent Knowledge* and a *Project Dependent Knowledge*. The former is then split into an *Ontology Layer* including entities, properties, relationships and rules, and, on top of this layer, into a *Deductive-Reasoning Layer*. *Project Independent Knowledge* has hitherto been subdivided into *Specialist Knowledge* and *Common Knowledge*. Altogether it can be viewed as a series of formalized handbooks directly and dynamically linked among themselves and to the ongoing design project.

In the specific *Project Dependent Knowledge* any actors through her/his own *Specialist Knowledge,* builds her/his own new entities (ontologies, relations, attributes and rules) relevant for the specific project.

At the lower level (or the *Instance Data level*) *data structures* are defined that make up both the specialist and the common design instances, depending on the specific design project. Instance Data consist essentially of the *values* of the attributes and of the relations among the entities defined in the upper levels by the actors, that are progressively specified in the course of the development of the design process together with the corresponding entities in the upper levels that can be modified at will (fig. 3).

Actors proceed to model the behaviour and verify the internal consistency of their own specialist instance using their own design methods and techniques with the help of their application software.

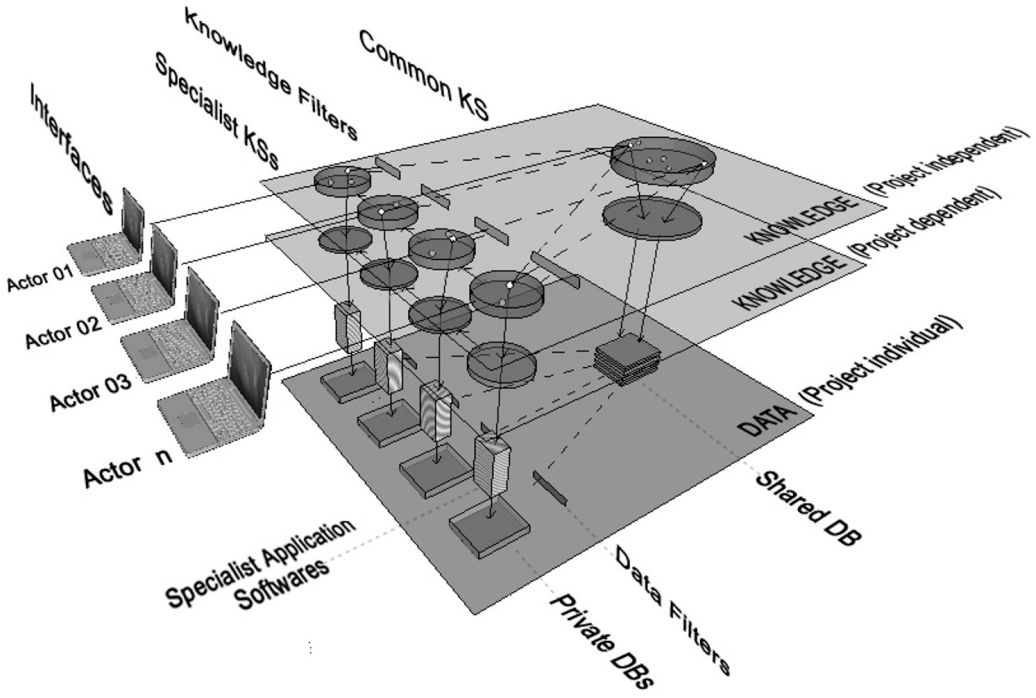


Figure 3. System for collaborative building design based on Knowledge Structures.

6. Quality issues in the design process

We address the issue of possible improvements to the quality of the design process, by leveraging on the knowledge structures presented above. Collaborative design has analogies in all areas, and these ideas might be of general interest.

The adoption of ontologies together with abstract representation layers allows the actors to define a complete break-down description of the artifact in terms of the physical entities that will be a part of the building. This is particularly effective in the early stages of the design, when most details have not yet been defined. The designer has the opportunity to state, by means of a formal definition: constraints of a general nature, optimization metrics, desired performances, or – generally speaking – any indicator or quality function (functions that take a dataset as input and return a quality value for each argument). During the design process each component of the building is defined and instantiated. When a preliminary project is available (of even earlier, if an appraisal can be evaluated), the value of these functions may be computed and constantly updated throughout the further evolution of the project, on any design variant, immediately showing the effects of design choices, instead of being calculated a posteriori.

The filtering mechanism – adjusted by any single actor according to the current situation – allow the actor to define and tune-up one or more "view"s of the project on a specific need; at this aim this view is, actually, a "report", tailored to specific requirements, dynamically and automatically updated during the progress of the project, largely reusable on different portions of the same project and/or subsequent projects.

The separation between project-dependent and project-independent knowledge structures is an interesting opportunity for process improvements. As the design activity proceed, (part of) the work done and stored in the Project Dependent Knowledge Base can be imported by each actor in his/her Project Independent Knowledge Base. Each actor is encouraged to play the role of a knowledge engineer in order to update the ontologies with the new definitions. Again, the existence of several abstraction layers in the current project, together with relations, constraints, and quality functions that have been defined and/or applied build up a library of reusable solutions to be used in new projects, with a different instantiation. The accumulated knowledge bases, possibly enlarged after each project, provide a base for a knowledge management, both for the individuals and for organizations involved in design activities.

7. Early Prototype Implementation and Future Developments

The paradigm proposed in this paper aims at improving design methodologies in the AEC industry by means of a novel approach to structuring and managing knowledge: this approach requires an adequate IT technology support. The latter is currently focused mainly on single-step or single-actor view points (standard CAD applications and suites), or on standard formats (IFC, BIM) which are quite complex and suitable only for machine-oriented data management.

Notably, the research focused on:

- exchanging information among the actors throughout the whole design process, starting from the early stages;
- adopting an ontology that could support the required representation of knowledge at various levels of abstraction, heterogeneous terminologies, semantic relationships;

- splitting knowledge used in the design process into Common / Specialist Knowledge and Project-Independent / Project-Dependent Knowledge.

The described Collaborative Working Environment has been implemented as a demonstrative prototype system, able to support a highly interactive collaborative design process among three specialist actors in the field of Architecture, Structural and Mechanical Engineering. We have proposed a framework architecture for a modular software platform supporting collaborative design. Some of the main features, partially implemented in the CoKAAD prototype system (Nanni & Santacaterina, 2009), are summarized below.

All the users are intended to use the same software installation, designed to be used by alternating online (connected) and offline (individual) work sessions. A client-server logical configuration has to be defined at the beginning of each online work session, designating a server; each actor has an initial option of synchronizing the offline work or importing the current state of the design activity.

The main tasks of the designated server are: accepting and managing the online collaborative work session (including connection control and concurrency management), managing the Overall Workspace and broadcasting each shared action to the designated actors. A fundamental task of the server is to record the operations carried out by the various actors in the Overall Workspace, maintaining a session log recording each single action, and a version history tree.

The user interface of a CoKAAD client is a desk with a collection of tools both for the individual work and for shared activity and communication with other actors; these tools include:

- standard collaborative environments (e.g., text and voice communication, file sharing)
- a free-hand drawing tool
- an installation of a CAD application with a plug-in intercepting all the relevant operations (currently we have interfaced Autodesk Architectural Desktop) for online teamwork
- a set of tools for the client-side control of the services provided by the server: connection control, access to session log and version history, with the possibility to run back and forth over the log and the version tree.

Additional background tasks of the client application are: intercepting meaningful actions in the Overall Workspace (to be forwarded to the server), applying filters to the incoming and outgoing flow of information, managing the Personal Workspace, and the individual view of the Overall Workspace.

The architecture of the framework presented here includes an engine for the representation and querying of ontologies. This is required in order to handle and combine the modular Structure of Knowledge, split into Common Knowledge, Specialist Knowledge (a distinct one for each actor), Project-Dependent Knowledge. The key task of concept matching may be performed at various levels of abstraction to activate a rule or a constraint and to clear terminology mismatches. This problem is tackled by means of deductive rules – the Reasoning Level above the Ontology Level.

The described Knowledge Structure has been tested by means of a use case study: a meta-design of a demonstrative hospital ward. Such a case study was chosen for the following reasons: it is a complex structure even in the case of reduced physical size, has similar requirements in all the EU countries and demands the contribution of numerous highly differentiated specialist skills that must be melded into an organic and balanced solution.

At present the implementation of such a demonstrative prototype system is under way, in order to support a highly interactive collaborative design processes among three specialist actors in the field of Architecture, Structural Engineering and Building Science (Chen, 2004). This implementation will make use of QuOnto (www.dis.uniroma1.it/~quonto/), developed in previous years at Sapienza University of Rome. This system, based on Description Logic, has proven to be computationally very efficient and robust enough to be used in productive environments with a million instances (Calvanese et al., 2008).

8. Conclusions and Expected Results in Building Collaborative Design

In this paper we propose a way of organizing knowledge in an integrated collaborative AEC design environment. We do not address many other issues concerning the design process and how it might be supported by a computing architecture. As an example, sharing information among actors can be done straightforwardly by using the Overall Design Workspace, or by implementing services that support query/answering among the individual installations – i.e., the Personal Workspaces.

In the early stages of the design activity, as well as any time an actor wants to open and explore a new alternative in the project, it is important that only a "draft" of a new solution may be specified. In order to avoid frustrating the creativity of an actor, it should be possible to express only some features of the alternative choice, i.e., a draft. It is important how the representation of such a Personal Design Instance, in a stage of draft, is handled:

- first it is represented only in the Personal Workspace
- then it is to be shared in the Overall Design Workspace.

The existence of different abstraction levels in the Knowledge model (with ontology mapping at all levels) allows an actor to specify and share with others a new alternative: this may possibly be inconsistent, missing many details, but cheap (for the proposer) and sufficient to express an emerging new idea. Note that this would be not possible with a full-featured data model requiring strong consistency.

By means of the Knowledge Structure presented, each actor is allowed to work using his/her/its own personal methods, algorithms, software and tools to represent and manage the complexity of his/her/its own instance as a solution of his/her/its own design problem. Beside this activity (which is what any designer makes with current tools today), the actor is supported by mapping among ontologies with the inferential engine, and the filtering mechanism. All actors have the added advantage that, although the other actors cannot enter their own 'Space' they are able to interact with the constraints/opportunities defined therein. Each human actors, helped by intelligent assistants, can define the instance data of his/her/its own design instance *by explicitly mapping* them with the conceptual entities they belong to, which are structured by relations and rules significant in their Specialist Knowledge. Actually the direct link between *concepts* and *instances* and the translation of both into *entities common to all the actors* makes it possible design collaboration, in the broad sense of the term, on the basis of a mutual comprehension and the sharing of the choices.

This evidently differs from what currently takes place in existing CAD systems in which the attribution of data to concepts is implicit, arbitrary and related to the subjective interpretative capacity of the various actors. For this very reason there does not (and cannot) exist in these

systems a knowledge or a data structure shared by all the actors, thereby ruling out any form of direct 'on-line' design collaboration. A comparison of the structure of existing CAD systems and the one described here shows how the principal innovation introduced by the latter consists in structuring, managing and sharing *knowledge***.**

The *expected results* of the proposed Knowledge Structure are the following ones.

The following are specific to the building design process:

- a more detailed investigation of the process logic, a comparison between the latter and the pathway envisaged by current legislation and regulations, an identification of critical aspects of the process in order to improve it;
- a better control and management of the design activity and the project's evolution to improve their quality/cost ratio, as far as the various phases (early conception, preliminary, detailed, constructive);
- a more competitive advantage of construction industry improving the efficacy and efficiency of the product-chain by means of asynchronous communication and knowledge sharing and expertise;
- a more coherent design logic: underlying reasons, intrinsic coherence, relations with the 'situation', iteration between actors and object, more conscious choices;
- a deeper exploration of the nature of collaborative design processes as a collaborative process on equal terms, from early conception, through manufacturing, to construction and maintenance;
- a competitive advantage to the production process, as an effective Collaborative Working Environment increases creativity and spreads innovation which play a key role in market success.

The following are specific to educational and social outcomes:

- an e-learning tool, a 'game', that can assist university students; it can make it easier to explore design solutions, acquire knowledge, be aware of design constraints in a complex field as it is architectural and building design process. The 'game' could be easily applied to other educational field;
- a general promotion of knowledge among professionals and workers, by the spreading of e-learning tools in industry, school, services' society.

To attain these goals a strong and 'collaborative' partnership is needed with European ICT industries, software houses, design firms, engineering firms and construction industries, along with universities and professional training.

Bibliography and References

Björk, B.C. (1999). Information Technology in construction: domain definition and research issues, *Computer Integrated Design and Construction*, **1**(1) 3-16.

Björk, B.C. (1992). A unified approach for modelling construction information, *Building and Environments*, **27**(2) 173-194.

Calvanese D., De Giacomo G., and Lenzerini M. (2008). Conjunctive query containment and answering under description logic constraints. *ACM Transactions on Computational Logic*, **9**(3):22.1–22.31, 2008.

Carrara, G. and Fioravanti, A. (2007). Collaboration, New Media, Design-An Integrated Environment for Supporting Collaboration in Building Design. In Pawlak, A., Sandkuhl, K. and Indrusiak, L.S. (eds.), *Coordination of Collaborative Engineering – State of the Art and Future Challenges*, GI-Edition Gesellschaft fur Informatik, e.V, Bonn: Köllen Druck + Verlag GmbH, pp. 143-160.

Carrara, G, Fioravanti A. (2004). How to construct an audience in Collaborative Design - The Relationship among which Actors in the Design Process. In B Rudiger, B Tournay and H Orbaek, (eds.), Architecture in the Network Society, 22nd eCAADe Congress, 15 - 18 September 2004, pp. 426-434.

Carrara, G., Fioravanti, A. and Nanni, U. (2004). Knowledge Sharing, not MetaKnowledge. How to join a collaborative design Process and safely share one's knowledge. In J. Pohl (ed.) *Intelligent Software System for the New Infrastructure*. San Luis Obispo (CA), Cal Poly, pp. 105-118.

Carrara G. and Fioravanti A. (2002). 'Private Space' and .Shared Space' Dialectics in Collaborative Architectural Design. In J. Pohl (ed.) *Collaborative Decision-Support Systems*, San Luis Obispo (CA), Cal Poly, pp. 27-44.

Carrara, G. and Fioravanti, A. (2001). A Theoretical Model of Shared Distributed Knowledge Bases for Collaborative Architectural Design. In J. Gero e K. Hori eds.. *Strategic Knowledge and Concept Formation III*, Sydney: Key Centre of Design Computing and Cognition - University of Sydney, pp. 129-143.

Carrara, G. and Kalay, Y.E. (1994). Past, present, future: process and Knowledge in Architectural Design. In G Carrara and Y.E. Kalay (eds.), *Knowledge-Based Computer-Aided Architectural Design*, Amsterdam: Elsevier Science Publishers B.V., pp. v-vii, 147-201, and 389-396.

Chen, P.-H., Cui L., Wan C., Yang Q., Ting S.K., Tiong R.L.K. (2004). Implementation of IFC-based web server for collaborative building design between architects and structural engineers, *Automation in Construction*, **14**(1) 115-128.

Cheng, M.-Y. (2008). Cross-organization process integration in design-build team, *Automation in Construction*, **17**(2) 151-162.

Fioravanti A. & Carrara, G. (2007). Philosophy and structure of a CWE-based Model of Building Design, [CD Rom]. In P. Cunningham and Miriam Cunningham (eds.), *eChallenges 2007,* Amsterdam: IOS Press, pp. 1-12.

Drakos, N. and Knox, R.E. (2004). You need More Than E-Mail to Share Tacit Knowledge. Stamford (CT) Gartner Research. Available from: http://www4.gartner.com/DisplayDocument?id=450075. [9 June 2004, accessed June 2010].

Gross, M.D., Yi-Luen Do, E., McCall, R., Citrin, W.V., Hamill, P., Warmack, A. and Kuczun, K.S. (1998). Collaboration and coordination in architectural design: approaches to computer mediated team work, *Automation in Construction*, **7**(6) 465-473.

Jeng, T.-S. and Eastman, C.M. (1998). A database architecture for design collaboration, *Automation in Construction*, **7**(6) 475-483.

Kolarevic, B., Schmitt, G., Hirschberg, U., Kurmann, D. and Johnson, B. (2000). An experiment in design collaboration, *Automation in Construction*, **9**(1) 73-81.

Kvan, T. (2000). Collaborative design: what is it?, Martens, B. (guest ed.), Special Issue eCAADe '97, *Automation in Construction*, **9**(4) 409-415.

Nanni, U. and Santacaterina, A. (2010). CoKAAD: a Framework for Collaborative Architectural Design. In G. Carrara, A. Fioravanti and Y. Kalay (eds.) *Collaborative Working Environments for Architectural Design*, pp. 225-235, Palombi Editori, Roma.

Nguyen, T.-H., Oloufa, A.A. and Nassar, K. (2005). Algorithms for automated deduction of topological information, *Automation in Construction*, **14**(1) 59-70.

Ugwu, O.O., Anuba C.J. and Thorpe A. (2005). Ontological foundation for agent support in constructability assessment of steel structure – a case study, *Automation in Construction*, **14**(1) 99-114.

van Leeuwen J.P. and van der Zee, A. (2005). Distributed object models for collaboration in the construction industry, *Automation in Construction*, **14**(4) 491-499.

Wix, J. (1997). ISO 10303 Part 106, BCCM (*Building Construction Core Model*) /T200 draft.

Intelligent Software for Ecological Building Design[1]

Jens Pohl[A], Hisham Assal[A], and Kym Jason Pohl[B]
[A]Collaborative Agent Design Research Center (CADRC)
California Polytechnic State University (Cal Poly)
[B]CDM Technologies, Inc.
San Luis Obispo, California, USA

Abstract

Building design is a complex process because of the number of elements and issues involved and the number of relationships that exist among them. Adding *sustainability* issues to the list increases the complexity of design by an order of magnitude. There is a need for computer assistance to manage the increased complexity of design and to provide intelligent collaboration in formulating acceptable design solutions. Software development technology today offers opportunities to design and build an intelligent software system environment that can serve as a reliable intelligent partner to the human designer.

In this paper the authors discuss the requirements for an intelligent software design environment, explain the major challenges in designing this environment, propose an architecture for an intelligent design support system for sustainable design and present the existing technologies that can be used to implement that architecture.

1. Introduction

Design is indeed a ubiquitous activity. In the physical world every artifact, whether it be a coffee maker, a miniature silicon sensor for invasive blood pressure monitoring, an automobile, or a building, is the result of some kind of design activity. However, design is concerned not only with the creation of artifacts. Any problem solving situation in which there exists an element of the unknown, such as lack of information or incomplete knowledge of the relationships among issues, involves an intellectual effort that can be categorized as design (Simon 1996).

Typically, design requires decisions to be made among several imperfect alternatives. It is in the nature of those decisions that designers will often find the need to supplement logical reasoning with intuitive feelings about the problem situation that can lead to creative solutions and new knowledge. As a rule such new knowledge cannot be logically deduced from the existing available knowledge and is validated only after the solution has been discovered and tested. In this respect design is not unlike the decision making activities that occur in a wide range of complex problem situations that have to be dealt with in many professional fields such as management, economics, medicine, law, transportation planning, and military command and control.

2. The Inherent Complexity of Building Design

Design is the core activity in the field of architecture. The design of even a relatively simple low-rise building can be a complex task involving critical issues related to macro and micro climatic

[1] An abbreviated version of this paper was presented as a keynote address at the 2nd International KES IDT'10 Conference, Inner Harbor, Baltimore, Maryland, USA, 28-30 July, 2010 and the full length paper will appear in a 2011 edition of the Intelligent Decision Technologies (IDT) Journal.

conditions, building loads and structural system selection, site planning, internal space layout, heating and cooling, ventilation, lighting, noise control and room acoustics, construction materials and finishes, security, privacy, construction duration and cost, labor and product availability, and aesthetics. Since many of these design issues tend to conflict in different ways, it is not just the number of issues involved but in particular the relationships among the issues that are the core cause of design complexity.

To come to terms with such a complex problem solving environment architects pursue an iterative path of analysis, synthesis, and evaluation that requires the design problem to be decomposed into multiple sub-problems (Pohl 2008). Typically, they will select what they consider to be the most important issues and analyze those largely in isolation from the other issues. The results of this analysis are then synthesized into narrow solutions, which are evaluated in the context of both the selected and the remaining issues. When the narrow solutions fail to adequately cater for some of the issues the entire analysis, synthesis, and evaluation cycle is repeated with the objective of generating better narrow solutions. This is a laboriously cyclic process that results in the progressive adaptation and refining of the narrow solutions into broader solutions until the designer is either satisfied with the result or has exhausted the available time and/or financial resources.

3. Increased Complexity of Ecological Design

Based on current and historical building construction and occupancy experience it is quite difficult to imagine the design and operation of a building that is not in some measure destructive to the natural environment. Typically: the site is graded to provide convenient vehicular access and suit the layout of the building and its immediate surroundings; the construction materials and components are produced from raw materials that are extracted from nature and consume a great deal of energy during their production; the materials and components are transported to the site consuming more energy in transit; on-site construction generates waste in terms of packaging material and the fabrication of footings, walls, floors, and roof; during the life span of the building energy is continuously consumed to maintain the internal spaces at a comfortable level and power the multiple appliances (e.g., lights, communication and entertainment devices, food preservation and preparation facilities, and security systems); despite some concerted recycling efforts much of the liquid and solid waste that is produced during the occupancy of the building is normally collected and either treated before discharge into nature or directly buried in landfills; and finally, at the end of the life span when the building is demolished most, if not all, of the construction materials and finishes are again buried in landfill sites.

Let us consider the other extreme, a building that has been designed on ecological principles and is operated as a largely self-sufficient micro-environment. Ecological design has been defined in broad terms as being in symbiotic harmony with nature (Van Der Ryn and Cowan 1996, Kibert 2005). This means that the building should integrate with nature in a manner that is compatible with the characteristics of natural ecosystems. In particular, it should be harmless to nature in its construction, utilization, and eventual demolition. The implementation of ecological design concepts in architecture has gained momentum over the past two decades with the increasing adoption of *sustainability* as a primary design criterion.

In the context of the built environment *sustainability* is the overarching concept that acknowledges the need to protect the natural environment for future generations[2]. It proposes that anything that we build today should be sustainable throughout its life span. Furthermore, at the end of its life span it should be amenable to deconstruction and the reuse of all of its materials in some form. Since the emergence of an energy crisis in the US in the early 1970s, due to an Arab-Israeli conflict, the emphasis has been placed on energy conservation during the life span of a structure. This must be viewed as a very small, first step in the quest for a sustainable built environment that is based on ecological design principles. For a building to meet the full intentions of *sustainability* it would need to:

- be constructed only of materials and products that are reusable in some form or another at the time of deconstruction of the building and, by implication, most of these materials would already contain recycled ingredients;
- be constructed of materials and products that used as little energy (i.e., embodied energy) as possible during their manufacture;
- be constructed of materials that are not subject to toxic off-gassing;
- be as close to energy self-sufficiency as possible subject to climatic and technology limitations;
- employ water harvesting, treatment and reuse strategies to reduce its freshwater draw to the smallest possible amount (i.e., about 10% of existing usage based on current predictions); and,
- incorporate a waste management system that is capable of recycling most, if not all, of the dry and wet waste produced in the building.

The overarching impact of such stringent sustainability-based design and occupancy requirements adds an order of magnitude of complexity to an already very complex and time consuming building design process. Whereas architects practicing in the 20th Century already had to deal with a host of often conflicting design issues ranging from space planning and three-dimensional modeling to structural and environmental system selection, 21st Century architects will have many more considerations added to their plate. For example, they will need to justify the use of every material, not only in respect to cost and serviceability but also based on embodied energy and potential toxicity parameters, as well as the ability to recycle the material. The need to minimize water usage will require the use of graywater with the necessary capture and recycling facilities. Most, if not all, of the energy used in a new residential building will most likely have to be captured on-site.

How will the architect be able to cope with the increasing complexity of the building design process under these exacting ecological design principles based on *sustainability* criteria? Clearly, this is not just a matter of academic preparation and experience, but will depend on the ability of the designer to apply sufficient technical depth and breadth to the development of the design solution. Such an ability will increasingly depend on the availability of an arsenal of readily accessible and seamlessly integrated design tools. What is required amounts to an intelligent design environment that seamlessly assists the designer in finding and gaining access to the required information, generating and evaluating narrow solutions on the basis of

[2] The Bruntland Report (1987) defined *sustainable development* as "… meeting the needs of the present without compromising the ability of future generations to meet their needs" (UN (1987); 'Our Common Future'; United Nations, World Commission on Environment and Development, A/42/427 Supplement 25, 4 August, New York, New York).

simulations, identifying and resolving conflicts as narrow solutions are merged into broader solutions, and continuously monitoring the progress of the overall design solution within a dynamically interactive and collaborative software environment.

4. Desirable Capabilities of an Intelligent Design Environment

Some importance is attached to the term *environment* in preference to the more conventional nomenclature that would refer to a related set of software components that are intended to interoperate as a *system*. The use of the term environment is intended to convey a level of integration of capabilities that is seamless and transparent to the user. In other words, while engaged in the design activity the designer should not be conscious of the underlying software and inter-process communication infrastructure that is necessary to support the operation of the environment. The objective is for the designer to be immersed in the design activity to the extent that both the automated capabilities operating mostly in background and the capabilities explicitly requested by the user at any particular time operating in foreground are an integral part of the process. Ideally, the designer should perceive the design process and the environment within which the design activity is being performed as being synonymous.

From a general point of view there are at least two overriding requirements for an intelligent computer-based design environment. The first requirement relates to the representation of information within the environment. The software must have some level of *understanding* of the information context that underlies the interactions of the human user with the environment. This is fundamental to any meaningful human-computer interaction that is akin to a partnership. The level to which this *understanding* can be elevated will largely determine the assistance capabilities and essentially the value of the software environment to the human designer.

The second requirement is related to the need for the designer to be able to collaborate. In a broad sense this includes not only the ability to interact with human users who play a role in the design process, such as members of the design team, specialist consultants, material and product vendors, contractors and subcontractors, the building owners and their representatives, and local building authorities, but also non-human sources of information and capabilities. All of these interactions between the designer, other human participants in the design process, data sources, and software-based problem solving capabilities, must be able to be performed seamlessly without the user having to be concerned about access protocols, data formats, or system interoperability issues.

While these overall requirements would at first sight appear to be utopian compared with the state of computer-based environments that exist today (2010), the technology needed for the creation of such environments has been rapidly emerging during the past decade and is now largely available. However, before addressing the technical software design aspects it will be necessary to delve more deeply into the functional requirements of the postulated intelligent design environment.

4.1 Emphasis on partnership

A desirable computer-aided design environment is one that assists and extends the capabilities of the human designer rather than replaces the human element. Human beings and computers are complementary in many respects. The strengths of human decision makers in the areas of conceptualization, intuition, and creativity are the weaknesses of the computer. Conversely, the strengths of the computer in computation speed, parallelism, accuracy, and the persistent storage of almost unlimited detailed information are human weaknesses. It therefore makes a great deal

of sense to view a computer-based design environment as a partnership between human and computer-based resources and capabilities.

This is not intended to suggest that the ability to automate functional sequences in the computer-based environment should be strictly confined to operations that are performed in response to user actions and requests. Apart from the monitoring of problem solving activities, the detection of conflicts, and the execution of evaluation, search and planning sequences, the computer-based environment should be able to undertake proactive tasks. The latter should include not only anticipation of the likely near-term need for data from sources that may be external to the design environment and need to be acquired by the environment, but also the exploration of alternative solution strategies that the environment considers promising even though the user may be currently pursuing another path.

In this partnership a high level of interaction between the designer and the computer-based design environment is a necessary feature. It provides opportunities for the designer to guide the environment in those areas of the decision-making process, such as conceptualization and intuition, where the skills of the user are likely to be far superior to those of the computer. Particularly prominent among these areas are conflict resolution and risk assessment. While it would be of considerable assistance to the designer to be alerted to conflicts and for the nature of the conflicts to be clearly identified, the resolution of such conflicts should not be automated but undertaken in collaboration with the designer.

It follows that the capabilities of the computer-based environment should be designed with the objective of assisting and complementing the user in a teaming role. Such tools are interactive by nature, capable of engaging in collaboration with the user to acquire additional information to help better understand the situation being analyzed. These tools are also able to provide insight into the reasoning processes that they are applying, thereby allowing the designer to gain confidence in their inferencing capabilities as well as make subtle adjustments in the logic being applied. The authors' past experience with multi-agent decision-support applications has shown that tools that are engineered for collaboration with each other and the human user provide opportunities for augmenting their capabilities through user interaction during execution (Pohl et al. 1997). It is therefore suggested that these kinds of tools better assist designers in dealing with the complexity of design. In other words, a collaborative approach affords the necessary visibility and agility to deal with the large number of considerations across a far reaching set of domains that characterizes the design activity.

4.2 Collaborative and distributed

Design or complex problem environments in general normally involve many parties that collaborate from widely distributed geographical locations and utilize information resources that are equally dispersed. A computer-based design environment can take advantage of the distributed participation by itself assuming a distributed architecture. Such an architecture typically consists of several components that can execute on more than one computer. Both the information flow among these components and the computing power required to support the system as a whole can be decentralized. This greatly reduces the potential for communication bottlenecks and increases the computation speed through parallelism.

Another advantage of the distributed approach is the ability to modify some components of the system while the system as a whole continues to operate with the remaining components. Similarly, the malfunction or complete failure of one component does not necessarily jeopardize the entire system. This is not so much a matter of redundancy, although the distributed

architecture lends itself to the provision of a high degree of redundancy, but rather a direct result of the physical independence of the components. While the components may be closely integrated from a logical point of view they can operate in their own autonomous physical environment.

4.3 An open architecture

The high degree of uncertainty that pervades complex problem environments, such as design, extends beyond the decision-making activity of the collaborating problem solvers to the configuration of the computer-based environment itself. The components of a design environment are likely to change over time, through modification, replacement, deletion, and extension. It should be possible to implement these changes in a seamless fashion through common application programming interfaces and shared resources. Service-Oriented Architecture (SOA) concepts align well with this principle as the functionality comprising the proposed design environment could be accommodated as a composition of discrete, self-contained software services with a very low degree of coupling between components.

4.4 Tools rather than solutions

The computer-based design environment should offer a set of tools rather than solutions to a predetermined set of problems. The indeterminate nature of design problems does not allow us to predict, with any degree of certainty, either the specific circumstances of a future problem situation or the precise terms of the solution. Under these circumstances it is far more constructive to provide tools that will extend the capabilities of the human designer in a highly interactive problem solving environment.

In this sense a tool is defined more broadly than a sequence of algorithms, heuristics or procedures that are applied largely on the direction of a user. Tools can be self-activating, be capable of at least semi-autonomous behavior, and cooperate with each other and users in employing and providing services.

4.5 Expressive internal representation

The ability of the computer-based environment to convey a sense of having some level of understanding of the meaning of the data and in particular the concepts being processed is the single most important prerequisite for a collaborative design environment (Assal et al. 2009). An expressive representation of the real world entities and concepts that define the problem space forms the basis of the interactions between the users and the design environment and, also, the degree of intelligence that can be embedded within its components. To the designer a building consists of real world entities such as rooms, walls, windows, and doors, as well as related concepts such as accessibility, energy conservation, and structural efficiency. Each of these notions has properties and relationships that determine their behavior under certain conditions. These semantic descriptors form the basis of collaboration among human problem solvers and are therefore likewise the fundamental subject matter being discussed in a computer-based design environment.

4.6 Embedded knowledge

The computer-based design environment should be a knowledge-based environment. In this context knowledge can be described as experience derived from observation and interpretation of past events or phenomena, and the application of methods to past situations. Knowledge-bases

capture this experience in the form of rules, case studies, standard practices, and typical descriptions of objects and object systems that can serve as prototypes. Problem solvers typically manipulate these prototypes or patterns through adaptation, refinement, mutation, analogy, and combination, as they apply them to the solution of current problems (Gero et al. 1988, Pohl 2008).

4.7 Decentralized decision-making

The computer-based design environment need not, and should not, exercise centralized control over the problem solving process. Much of the design activity will be localized and performed in parallel involving the collaboration of different members of the design team. In this regard building design is neither a rigidly controlled nor a strongly disciplined activity, but more aptly described as a process of information seeking, discovery, and subsequent processing. For example, intelligent and dynamically interactive design tools that are responsible for pursuing the interests of real world objects, such as spaces and other building elements (Pohl 1996) and management personnel in commercial and industrial applications (Pan and Tenenbaum 1991), can achieve many of their objectives through employing services and engaging in negotiations that involve only a few nodes of the design environment. This greatly reduces the propensity for the formation of communication bottlenecks and at the same time increases the amount of parallel activity in the computer-based environment.

The ability to combine in a computer-based design environment many types of semi-autonomous and autonomous components (i.e., agents), representing a wide range of interests and incorporating different kinds of knowledge and capabilities, provides the environment with a great deal of versatility and potential for problem solving to occur simultaneously at several levels of granularity. This is similar to human problem solving teams in which individual team members work concurrently on different aspects of the problem and communicate in pairs and small groups as they gather information and explore sub-problems.

4.8 Emphasis on conflict identification

The capabilities of the computer-based design environment should not be bound by the ultimate goal of the automated resolution of conflicts. Rather, the capabilities of the computing environment should support the identification of the conflict, presenting the human designer with as much of the related context as possible. This notion gains in importance as the level of complexity of the design problem increases. The resolution of even mundane conflicts can provide subtle opportunities for advancing toward design solution objectives. These opportunities are more likely to be recognized by a human designer than a computer-based agent. The identification of conflicts is by no means a trivial undertaking. It includes not only the ability to recognize that a conflict actually exists, but also the determination of the kind of conflict and the relationships and related context that describe the conflict and what considerations appear relevant to its resolution. The automatic tracing of these relationships may produce more progress toward a solution than the automatic resolution of the conflict itself.

4.9 Adaptability and agility

Traditionally, software tools categorized as intelligent were engineered for specific scenarios. Consequently, the successful application of these tools depended largely on the degree to which the characteristics of a particular problem component aligned with situations that the tool had been design for. This rigidity has tended to prove quite problematic when these tools were

applied to even slight variations of the scenarios that they had been developed or trained for.

In contrast, what the experience of the authors has shown is that intelligent tools not only need to support variation, but that these tools should be engineered with such adaptation as a core criterion. Much of this ability to effectively deal with variation is due to the ability of these tools to decompose complex problems into much more manageable components without losing the relationships that tie the components together. To accomplish this, the reasoning capabilities of the tools can be organized as discrete fragments of logic capable of addressing smaller components of the larger problem. If these components are described within an expressive, relationship-rich representation then the connections between the decomposed components are maintained automatically. The effects of addressing each individual component are automatically propagated across the entire expanse of the problem space due to the extensive set of relationships represented within the model that retains their connections and context. The result is a problem solving tool that is agile in its ability to effectively adjust to the variable nature of the evolving design solution.

4.10 The human-computer interface

The importance of a high degree of interaction between the human members of the design team and the various intelligent components of the computer-based design environment is integral to most of the principles and requirements described above. This interaction is fundamentally facilitated by the information-centric representation core of the environment through which the interacting software components are able to maintain some level of understanding of the current context of the design activity. However, there are other aspects of the user-interface that must be provided in support of the human-computer interactions. These include two-dimensional and three-dimensional graphical representation capabilities, explanation facilities, and a context-sensitive help system with semantic search support.

At a minimum the graphical capabilities must be powerful enough to include the accurate representation of the analysis results of the progressively evolving design solution in terms of the environmental factors that are involved in building design, such as: shadows based on sun path projections; daylighting and artificial lighting simulations within the building interior to the extent that adverse conditions such as glare can be readily perceived by the human designer; structural behavior based on the simulation of static dead and live loads, as well as dynamic wind and seismic loads; animated air movement and heat flow simulations; and, pedestrian traffic visualization. Technology permitting, the ultimate aim of the design environment should be to provide a virtual reality user-interface that allows the human designer to become fully immersed in the physical and emotional aspects of the design experience.

Explanation facilities: The authors' experience with decision-support systems over the past two decades has lent credence to the supposition that the need for the proposed design environment to be able to explain how it arrived at certain conclusions increases with the sophistication of the inferencing capabilities embedded in the software environment. At the very least, the intelligent components of the environment should be able to explain their results and methods of analysis. In this regard retrospective reasoning that is capable of providing answers to *what*, *how*, and *why* questions is the most common type of explanation facility available in multi-agent systems. A *what* question requires the explanation or definition of a fact. For example, in the context of architectural design the user may ask: *What are the characteristics of the window in the north wall of the conference room?* In the past, expert system methodologies based on *format templates* would have allowed the appropriate answer to be collected simply through template

values when a match is made with the facts (i.e., window, north, wall, conference) contained in the question (Myers et al. 1993). Today, with the application of ontology-based reasoning capabilities more powerful and direct methods based on the ability of an ontology to represent concepts are available. A *how* question requires an analysis of the sequence of inferences or reasoning that produced the fact. Continuing with the above example, the designer may ask: *How can the intrusion of external noise into the conference room be mitigated?* The answer would require a sequence of inferences by the Noise Agent. This sequence can be preserved and presented to the designer.

Why questions are more complicated. They require reference to the sequence of goals that have driven the progression of inferences (Ellis 1989). In large collaborative systems many agents may have contributed to the inference sequence and will need to participate in the formulation of the answer. This third level of explanation, which requires a summary of justification components, has received considerable attention over the past 30 years. For example: text summary systems such as Frump (Dejong 1982) and Scisor (Jacobs and Rau 1988); fast categorization techniques such as Construe (Hayes and Weinstein 1991); grammatical inference (Fu and Booth 1975) that allows inductive operators to be applied over the sequences of statements produced from successive justifications (Michalski 1983); explanation-based learning (Mitchell et al. 1991); and, case-based reasoning (Shank 1990 and 1991).

Semantic search facilities: While existing computer-aided design systems typically support only factual searches, the proposed intelligent design environment should provide semantic search capabilities that can deal with inexact queries. Due to the complexity of the problem space the designers will not always know exactly what information they require. Often they can define only in conceptual terms the kind of information that they are seeking. Also, they would like their query to be automatically broadened with a view to discovering additional information that may be relevant to their current problem solving focus.

The desirability of the design environment to be able to deal with inexact search requests warrants further discussion. A flexible query capability, such as the human brain, can generate best guesses and a degree of confidence for how well the available information matches the query. For example, let us assume that the designer is searching for a window unit of something like the *double-hung* window type. The flexible query facility would presumably include a *something like or similar to* operator capable of matching in a partial sense. Windows that have a movable part are something like the *double-hung* window type. Windows that have their movable part in the vertical direction are more like *double-hung* than windows that have their movable part in the horizontal direction. Windows that open by rotation are even less like *double-hung* than windows that are simply fixed. In other words each of the *something like* information items would be validated by a degree of match qualification.

However, the capabilities of the proposed intelligent design environment should exceed the flexible query capabilities described above. It should also include the ability to automatically formulate hypotheses. The ability to search for *something like* is only a starting point, given that the designer has just inserted a window in the evolving design solution it would be useful for the design environment to automatically search for building types with similar window configurations. This will require the capability to automatically search for vaguely or conceptually related information. For example, if the design focus is currently on the window arrangement of a cafeteria space, the environment should be able to *discover* other spatial design solutions that are conducive to a relaxed eating atmosphere and perhaps provide a more appropriate window arrangement than the current solution.

5. The Technical Approach

The desired capabilities of the proposed intelligent design environment outlined in the previous section call for a distributed system architecture that can be accessed from any physical location, is highly flexile, and totally transparent to the human user. In particular, the user must be shielded from the many protocols and data and content exchange transformations that will be required to access capabilities and maintain seamless interoperability among those capabilities. Any member of the design team, once authenticated during the single sign-on point of entry, should be able to access those capabilities (e.g., intelligent design tools and data sources) that are included in the authentication certificate. The focus of the designer should not be on systems, as it still is mostly today, but on the capabilities or *services* that the computer-based environment can provide.

The notion of *services* is well established. Everywhere we see countless examples of tasks being performed by a combination of services, which are able to interoperate in a manner that results in the achievement of a desired objective. Typically, each of these services is not only *recomposable* but also sufficiently *decoupled* from the final objective to be useful for the performance of several somewhat similar tasks that may lead to quite different results. For example, a common knife can be used in the kitchen for preparing vegetables, or for peeling an orange, or for physical combat, or as a makeshift screwdriver. In each case the service provided by the knife is only one of the services that are required to complete the task. Clearly, the ability to design and implement a complex process through the application of many specialized services in a particular sequence has been responsible for most of mankind's achievements in the physical world.

5.1 Service-oriented architecture (SOA)

In the software domain these same concepts have gradually led to the adoption of Service-Oriented Architecture (SOA) principles. While SOA is by no means a new concept in the software industry it was not until Web services became available that these concepts could be readily implemented (Erl 2008, Brown 2008). In the broadest sense SOA is a software framework for computational resources to provide services to customers, such as other services or users. The Organization for the Advancement of Structured Information (OASIS)[3] defines SOA as a *"... paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains"* and *"...provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects with measurable preconditions and expectations"*. This definition underscores the fundamental intent that is embodied in the SOA paradigm, namely *flexibility*. To be as flexible as possible a SOA environment is highly modular, platform independent, compliant with standards, and incorporates mechanisms for identifying, categorizing, provisioning, delivering, and monitoring services.

The principal components of a conceptual SOA implementation scheme (Figure 1) include a Enterprise Service Bus (ESB), one or more portals to external clients with single sign-on facilities, and the enterprise services that facilitate the ability of the user community to perform its operational tasks.

[3] OASIS is an international organization that produces standards. It was formed in 1993 under the name of SGML Open and changed its name to OASIS in 1998 in response to the changing focus from SGML (Standard Generalized Markup Language) to XML (Extensible Markup Language) related standards.

Enterprise Service Bus (ESB): The concept of an Enterprise Service Bus (ESB) greatly facilitates a SOA implementation by providing specifications for the coherent management of services. The ESB provides the communication bridge that facilitates the exchange of messages among services, although the services do not necessarily know anything about each other. According to Erl (2008) ESB specifications typically define the following kinds of message management capabilities:

- *Routing:* The ability to channel a service request to a particular service provider based on some routing criteria (e.g., static or deterministic, content-based, policy-based, rule-based).
- *Protocol Transformation:* The ability to seamlessly transform the sender's message protocol to the receiver's message protocol.
- *Message Transformation:* The ability to convert the structure and format of a message to match the requirements of the receiver.
- *Message Enhancement:* The ability to modify or add to a sender's message to match the content expectations of the receiver.
- *Service Mapping:* The ability to translate a logical business service request into the corresponding physical implementation by providing the location and binding information of the service provider.
- *Message Processing:* The ability to accept a service request and ensure delivery of either the message of a service provider or an error message back to the sender. Requires a queuing capability to prevent the loss of messages.
- *Process Choreography and Orchestration:* The ability to manage multiple services to coordinate a single business service request (i.e., choreograph), including the implementation (i.e., orchestrate). An ESB may utilize a Business Process Execution Language (BPEL) to facilitate the choreographing.
- *Transaction Management:* The ability to manage a service request that involves multiple service providers, so that each service provider can process its portion of the request without regard to the other parts of the request.
- *Access Control and Security:* The ability to provide some level of access control to protect enterprise services from unauthorized messages.

There are quite a number of commercial off-the-shelf ESB implementations that satisfy these specifications to varying degrees. A full ESB implementation would include four distinct components (Figure 2): Mediator; Service Registry; Choreographer; and, Rules Engine. The Mediator serves as the entry point for all messages and has by far the largest number of message management responsibilities. It is responsible for routing, communication, message transformation, message enhancement, protocol transformation, message processing, error handling, service orchestration, transaction management, and access control (security).

The Service Registry provides the service mapping information (i.e., the location and binding of each service) to the Mediator. The Choreographer is responsible for the coordination of complex business processes that require the participation of multiple service providers. In some ESB implementations the Choreographer may also serve as an entry point to the ESB. In that case it assumes the additional responsibilities of message processing, transaction management, and access control (security). The Rules Engine provides the logic that is required for the routing,

transformation and enhancement of messages. Clearly, the presence of such an engine in combination with an inferencing capability provides a great deal of scope for adding higher levels of intelligence to an ESB implementation.

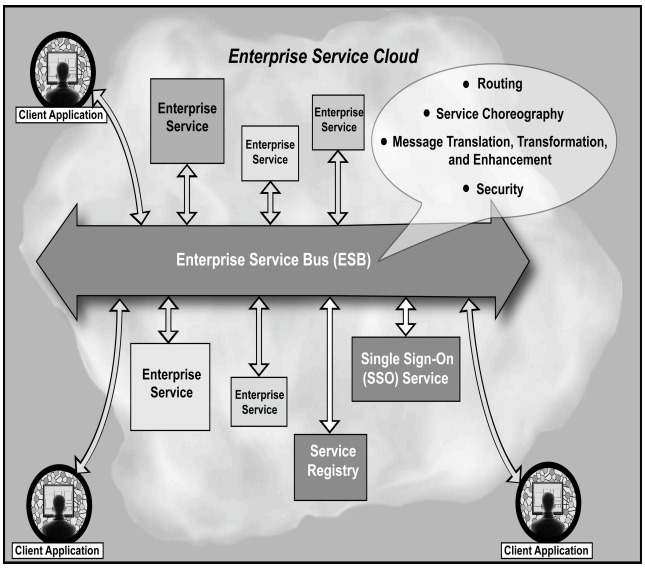


Figure 1: Principal SOA components

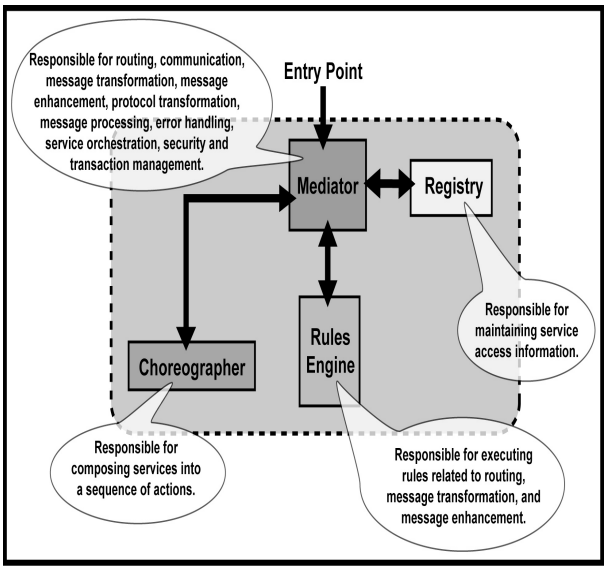


Figure 2: Principal ESB components

5.2 Information-centric representation

The methods and procedures that designers utilize to solve design problems rely heavily on their ability to identify, understand and manipulate objects. In this respect, objects are complex symbols that convey meaning by virtue of the explicit and implicit context information that they encapsulate within their domain. For example, architects develop design solutions by reasoning about neighborhoods, sites, buildings, floors, spaces, walls, windows, doors, and so on. Each of these objects encapsulates knowledge about its own nature, its relationships with other objects, its behavior within a given environment, what it requires to meet its own performance objectives, and how it might be manipulated by the designer within a given design problem scenario. This knowledge is contained in the various representational forms of the object as factual data, algorithms, rules, exemplar solutions, and prototypes (Pohl 2008, 59-62).

It is therefore apparent that a critical requirement for effective human-computer interaction in the proposed intelligent design environment is the appropriate representation of the evolving design solution model. This can be accomplished utilizing an *ontology*. The term ontology is loosely used to describe an information structure that is rich in relationships and provides a virtual representation of some real world environment. The elements of an ontology include objects and their characteristics, different kinds of relationships among objects, and the concept of inheritance (Assal et al. 2009). While an ontology is expressed in object-oriented terms, it is more than an object model. It is designed to describe the entities, concepts, and related semantics of some subject matter domain. Software that incorporates an internal information model, such as an ontology, is often referred to as *information-centric* software. The information model is a virtual representation of the real world domain under consideration and is designed to provide adequate *context* for software agents (typically rule-based) to reason about the current state of the virtual environment.

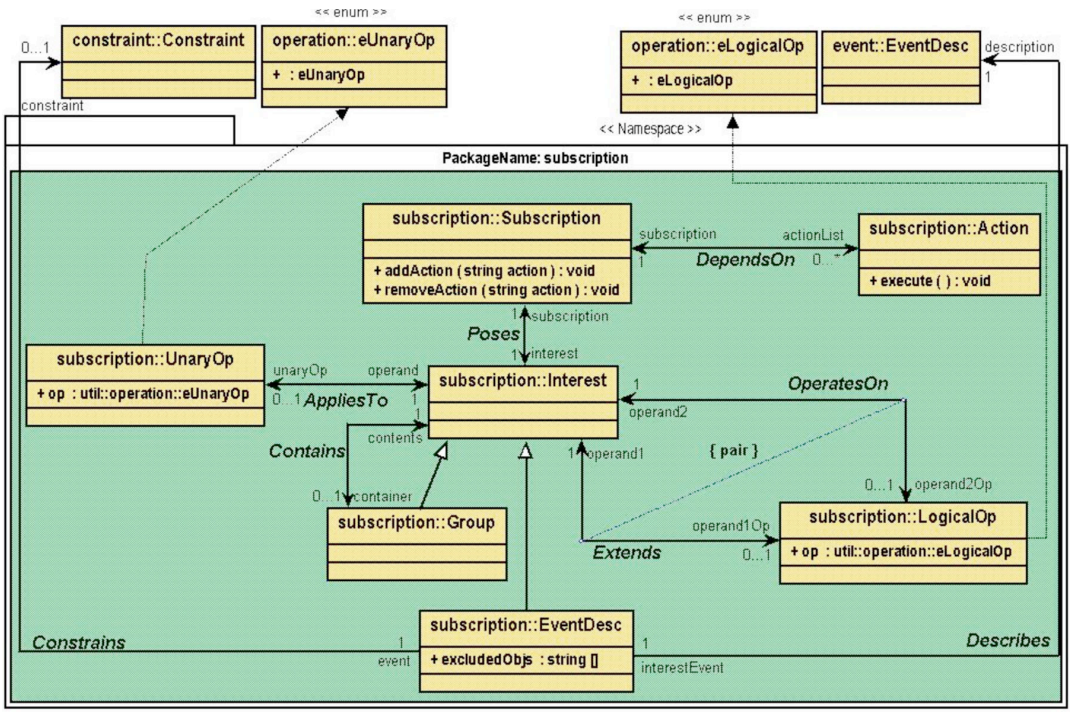


Figure 3: Typical subscription service domain ontology

Within a SOA-based system environment the various information-centric tools that are available to the designer will exist as an integrated collection of clients (i.e., users of the ontology), typically referred to as *services*. These services can communicate directly or indirectly via message translation, in terms of the real world objects and relationships that represent the contextual framework of the evolving design solution. To reduce the amount of work (i.e., computation) that the computer has to accomplish and to minimize the volume of information that has to be transmitted within the system, two strategies can be readily implemented. First, since the services involved in a particular collaboration are stateful in nature (i.e., they retain a working knowledge of the various aspects of the evolving design solution that they are concerned with) only the changes in information need to be communicated. For example, an agent that is monitoring the layout of spaces during the design of a building may have an extensive set of information concerns or interests relating to various aspects of the evolving design solution. These interests will likely include the location, geometric parameters and functional characteristics of a particular space. If the designer changes the locations of this space then only that aspect should be transmitted to interested parties.

Second, to further economize on communication traffic as well as increase the timeliness and efficiency with which components (i.e., agents, etc.) interact, an asynchronous notification facility (i.e., subscription service) can be employed where parties can indicate their respective information interests. When entries with such subscription profiles are satisfied, respective users are asynchronously notified allowing them to take whatever action they see fit. By allowing relevant information to be automatically *pushed* to interested parties, the subscription service obviates the need for repetitive queries and thereby greatly reduces the amount of work the computer has to perform.

The basic components comprising a subscription service architecture view clients as sharable, collaborative objects. Accordingly the subscription service is presented to clients in the form of

subscription objects that can be instantiated. These subscription objects embody the same set of qualities as any other object and can be represented in the underlying ontology as a subscription/notification domain (Figure 3). To invoke the subscription service a client may either instantiate a subscription object or instead chose to utilize an existing subscription registered by another client. Regardless of method, during this process the subscriber also associates a local action object to the subscription, to initiate the action that the client wishes to have executed upon subscription satisfaction.

5.3 Design tools with collaborative agents

On the assumption of an information-centric software architecture that incorporates an ontology-based high level representation of the design problem context, the intelligence of the proposed design environment will be largely contributed by the design tools that are available to the human designer. Most of these design tools will be in the form of invocable services or self-initiating agents. There is a behavioral distinction between services and agents. Services are invoked to perform a discrete activity, returning to their original inactive state after the activity has been completed. Agents on the other hand may be active on a continuous basis, taking the initiative opportunistically whenever they determine that the situation warrants an action. Often these agent actions will invoke services.

There are many types of software agents, ranging from those that emulate symbolic reasoning by processing rules, to highly mathematical pattern matching neural networks, genetic algorithms, and particle swarm optimization techniques. While all of these have capabilities that are applicable to an intelligent design environment, only symbolic reasoning agents that can interact directly with the ontology-based design context model will be discussed in this paper. For these rule-based agents the reasoning process relies heavily on the rich representation of entities and concepts provided by the ontology.

In general terms software agents with symbolic reasoning capabilities may be defined as tools that are situated, autonomous, and flexible (Wooldridge et al. 1999, Wooldridge 1997). They are situated since they receive a continuous flow of operational information generated by the activities within and peripheral to the problem domain environment, and perform acts that may change that environment (e.g., creating alerts, making suggestions, and formulating recommendations). Agent tools are autonomous because they act without the direct intervention of human operators, even though they allow the latter to interact with them at any time. In respect to flexibility, agent tools possess the three qualities that define flexibility within the context of the above definition. They are responsive, since they perceive their environment through an internal information model (i.e., ontology) that describes some of the entities and concepts that exist in the real world environment. They are proactive because they can take the initiative in making suggestions or recommendations. They are social, since they can collaborate with other agents or human users, when appropriate, to complete their own problem solving and to help others with their activities.

One important aspect of autonomy in agent applications is the ability of agents to perform tasks whenever such actions may be appropriate. This requires agents to be opportunistic, or continuously looking for an opportunity to execute. In this context opportunity is typically defined by the existence of sufficient information. For example, as the location of a particular space is defined by the designer within the evolving floor plan, several agents may become involved automatically to undertake analyses (e.g., thermal, lighting, acoustics) appropriate to their capability domains.

Planning Agents: Planning is a reasoning activity about the resources and actions to fulfill a given task. Planning Agents are complex agents that reason about the problem state and produce a plan based on the current state of the design in conjunction with the applicable constraints and objectives. This planning process involves matching the latter with the available resources to produce a course of action that will satisfy the desired objectives. The complexity of the process can be reduced by distributing the basic planning tasks among a set of agents, as follows: identify the constraints and objectives; identify the available resources; note the unavailability of resources; identify the available set of actions or characteristics; and, generate a plan for satisfying the objectives.

Plan or solution generation is the actual planning activity in the above list of tasks. Many planning systems use specialized search algorithms to generate plans according to given criteria (Blum and Furst 1997). Re-planning, which is also commonly referred to as continual planning, involves the re-evaluation of parts of an existing plan or solution because of a change in the information that has been used in the creation of that plan. This is a common situation in architectural design, where the designer is continuously adapting the evolving design solution during the iterative analysis-synthesis-evaluation cycle (Pohl 2008, 47-52).

Some planning systems take advantage of the feedback obtained from the monitoring and execution of plans to add to their knowledge by employing learning techniques, such as explanation-based learning, partial evaluation, experimentation, automatic abstraction, mixed-initiative planning, and case-based reasoning. There are several approaches to learning in agents, including reinforcement learning, classifier systems, and isolated concurrent learning. Learning techniques also enhance the communication ability of agents (Sen et al. 1994, Veloso et al. 1995).

Service Agents: Agents that are designed to be knowledgeable in a specific domain, and perform planning or assessment tasks in partnership with other agents (i.e., human agents or software agents) are often referred to as Service Agents (Durfee 1988, Durfee and Montgomery 1990, Pohl et al. 1997). The manner in which they participate in the decision-making activities depends on the nature of the situation. Service Agents can be designed to react to changes in the problem state spontaneously through their ability to monitor information changes and respond opportunistically. They should be able to generate queries dynamically and access resources automatically whenever the need arises.

In the proposed intelligent design environment both Service and Planning Agents will constitute the principal design tools by providing analysis, solution generation and evaluation capabilities for the full range of knowledge domains that impact an ecologically based design solution, namely: site analysis; building orientation; space layout optimization; structural system selection; deconstructability assessment; thermal design determinates; passive solar system analysis; mechanical heating, ventilating and air-conditioning solution generation and evaluation; daylighting and artificial lighting design; alternative energy analysis and solar system alternatives; room acoustics and noise insulation; building hydrology analyses; closed-loop material selection; embodied energy analysis; waste disposal and recycling; life cycle cost analysis; construction cost estimation; and so on.

What is of particular significance is that unlike the manual design process, which requires these related design factors to be considered in an essentially sequential manner, the various agents will be able to operate in parallel in the proposed design environment. Furthermore, the ability of the agents to collaborate will allow the relationships among the different knowledge domains to

be pursued dynamically. Since the complexity of the building design activity is due to the large number of relationships among the domains, the proposed design environment embodies the potential for dealing with a highly complex problem situation in a holistic manner.

Mentor Agents: A Mentor Agent is a type of agent that is based on the agentification of the information entities and concepts that are intrinsic to the nature of each application. In the proposed design environment these are the primary building elements and concepts that the architect reasons about and that constitute the foundations of the internal representation (i.e., ontology) of the problem situation within an information-centric software system (Pohl 1996). For example, a Mentor Agent may attend to the needs of a specific building space (i.e., an entity) or pursue energy conservation objectives (i.e., a concept) that govern the entire design solution. The concept of Mentor Agents brings several potential benefits.

First, it increases the granularity of the active participants in the problem solving process. As agents with collaboration capabilities, agentified design elements can pursue their own objectives and perform a significant amount of local problem solving without repeatedly impacting the communication and coordination facilities utilized by the higher level components of the distributed system. Typically, a Mentor Agent is equipped with communication capabilities, process management capabilities, information about its own nature, global objectives, and some focused problem solving tools.

Second, the ability of Mentor Agents to employ services greatly increases the potential for concurrent activities. Multiple Mentor Agents can request the same or different services simultaneously. If necessary, Service Agents responding to multiple service requests can temporarily clone themselves so that the requests can be processed in parallel. Third, groups of Mentor Agents can negotiate among themselves in the case of matters that do not directly affect other higher level components or as a means of developing alternatives for consideration by higher level components.

Fourth, by virtue of their communication facilities Mentor Agents are able to maintain their relationships to other aspects of the evolving design solution. In this respect they are the product of *decentralization* rather than *decomposition*. In other words, the concept of Mentor Agents overcomes one of the most serious deficiencies of the rationalistic approach to problem solving; namely, the dilution and loss of relationships that occurs when a complex problem is decomposed into sub-problems. In fact, the relationships are greatly strengthened because they become active communication channels that can be dynamically created and terminated in response to the changing state of the problem situation.

It may not be desirable to elevate all domain elements to agent status. Elements that are subservient to other elements in most respects are unlikely to gain much from the capabilities of being able to operate as an active agent. For example, in architectural design, elements representing building spaces play a fundamental role from the earliest stages of the design process. Windows, on the other hand, are largely subservient both in terms of function and impact on the space in which they exist.

In the realm of building design it would seem desirable to implement building spaces as agents. Since Mentor Agents have communication capabilities a conference room *Space Agent*, for example, would be able to collaborate with other agents such as Service and Planning Agents. If the conference room *Space Agent* is interested in determining where it is located in respect to any surrounding sources of noise it could invoke the services of a *Noise Agent* to identify any relevant noise sources. This example illustrates two distinct benefits: only the most necessary

computation has been performed; and, the information that forms part of the fundamental description of the results can be held anywhere in the system (as long as it is available to any other authorized agent). Second, by distributing the collaborating parties, as well as the information that is generated as a result of the servicing of the requests, the communications involved with both the current interactions and any future use of the relevant information have been likewise distributed. Accordingly, the potential for the occurrence of a communication bottleneck has been effectively reduced.

Agent collaboration and conflict management: In previous multi-agent design and military decision-support systems developed by the authors (ICADS 1991, AEDOT 1992, Nibecker et al. 2007, Diaz et al. 2006) conflicts arose when agents either disagreed among themselves or with a decision made by the designer. For example, the placement of a window in a particular space might provoke the latter type of conflict. If the designer places the window in the west wall of a conference room and a loud noise source such as a freeway runs parallel to the west boundary of the site, then the *Noise Agent* (a Service Agent) would insist on the removal of the window. The designer is able to resolve the conflict by relocating or deleting the window or, alternatively, may decide to overrule the Service Agent altogether. The conference room, as a passive entity, is involved in the conflict resolution process only as an information source that is used by the Service Agent in its deliberations (Pohl and Myers 1994). In other words, while the validation of the design decision is entirely dependent on the knowledge encapsulated in the informational entity the latter is unable to actively participate in the determination of its own destiny.

The situation is somewhat analogous to a scenario common in real life when one or more persons feel compelled to make decisions for another person, although the latter might be more competent to make those decisions. The outcome is often unsatisfactory because the decision makers tend to use general domain information where they lack specific knowledge of the other person. In other words, the individuality of the problem situation has been usurped by the application of generalizations and, as a result, the quality of the decisions that have been reached are likely to be compromised.

In the example of the window in the west wall of the conference room, if the conference room is a *Space* agent then much of the decision-making can be localized within the knowledge domain of the agent. As soon as the window has been placed in the wall by the designer the conference room *Space Agent* could pose two specific questions to the appropriate Service Agents (i.e., in this example the *Noise Agent* and the *Lighting Agent*): *What is the expected background noise level in the room due to the window?* and *What is the spatial distribution of daylight admitted through the window?* The answers to these questions can be compared by the conference room *Space Agent* directly to what it knows about its own acoustic and lighting needs. The development of alternative strategies for resolving the noise problem can now take place within the context of all of the information in the conference room *Space Agent's* knowledge domain. For example, the possibility of' relocating itself to a quieter wing of the building can be explored by the agent (with or without the active collaboration of the designer) as a direct consequence of its own deliberations.

There is another kind of conflict resolution scenario that becomes possible with the availability of Mentor Agents. An agent may develop a solution to a sub-problem in its own domain that redirects the entire design solution. In the conference room example the *Space* agent may resolve the noise control problem by adopting an expensive window unit (e.g., triple glazing) solution, and then continue to search for a more effective solution as the design solution continues to

evolve. The search may continue into subsequent stages of the design process, during which the conference room might progressively be governed by a Mentor Agent representing the entire floor or even the building as a whole. These higher level agents may now impose certain conditions on the *Space* agent for the greater good of the larger community. However, the *Space* agent, persevering in its search finally comes up with a method of noise control that utilizes a novel type of wall construction in combination with background masking sound. The proposed wall construction may even be contrary, yet still compatible, to that adopted for the external west wall of the building by both the *Floor* and *Building Agents*.

First, it is significant that this alternative solution has been found at all. If the conference room had been a passive data object there would not have been any desire on the part of the system to pursue the problem after the initial conflict resolution. Second, having found the alternative the conference room *Space Agent* is able to communicate its proposal and have the noise control issue reconsidered. It could engage in a discourse with, in order of authority, the *Floor Agent* and the *Building Agent*. At each of the agent levels there is the opportunity for wider consultation and interaction with the designer. Finally, if the proposal has been rejected at all higher agent levels, the conference room *Space Agent* may appeal directly to the designer. The designer has several alternative courses of actions available: also reject the proposal; require one or more of the higher level agents to explain their ruling; reset certain parameters that allow the higher level agents to reconsider their ruling; overrule the higher level agents and accept the proposal; or, capture the current state of the design solution as a recoverable snapshot and use the *Space Agent's* proposal as the basis for the exploration of an alternative solution path.

5.4 System architecture

The proposed system consists of a number of components in a SOA-based environment, and a client application that serves as the user-interface for all of the user-service communications (Figure 4).

The system components include:

- A client application with support for computer-aided drawing (CAD) capabilities and a Building Information Model (BIM) interface. The use of BIM captures the design information in a standard way, which can be communicated to other system components. A BIM model representation is typically in Extensible Markup Language (XML) format, which supports the hierarchical structure of design elements. The client application is the only user-interface in the system. It provides the user with tools to access the other services and presents the information generated by services (i.e., service results) within the CAD application. The client application also includes a Business Process Management (BPM) component to allow the user to describe a collaborative workflow, which may involve other human users (e.g., external structural consultant) and system services. The BPM component takes a user description of a process and hands it to the SOA-based environment, namely the Enterprise Service Bus (ESB), for execution. The client application also displays any information received from the services as the result of analysis, recommendations, or warnings.
- A CAD service, which is responsible for communicating between the CAD environment and the ontology environment.

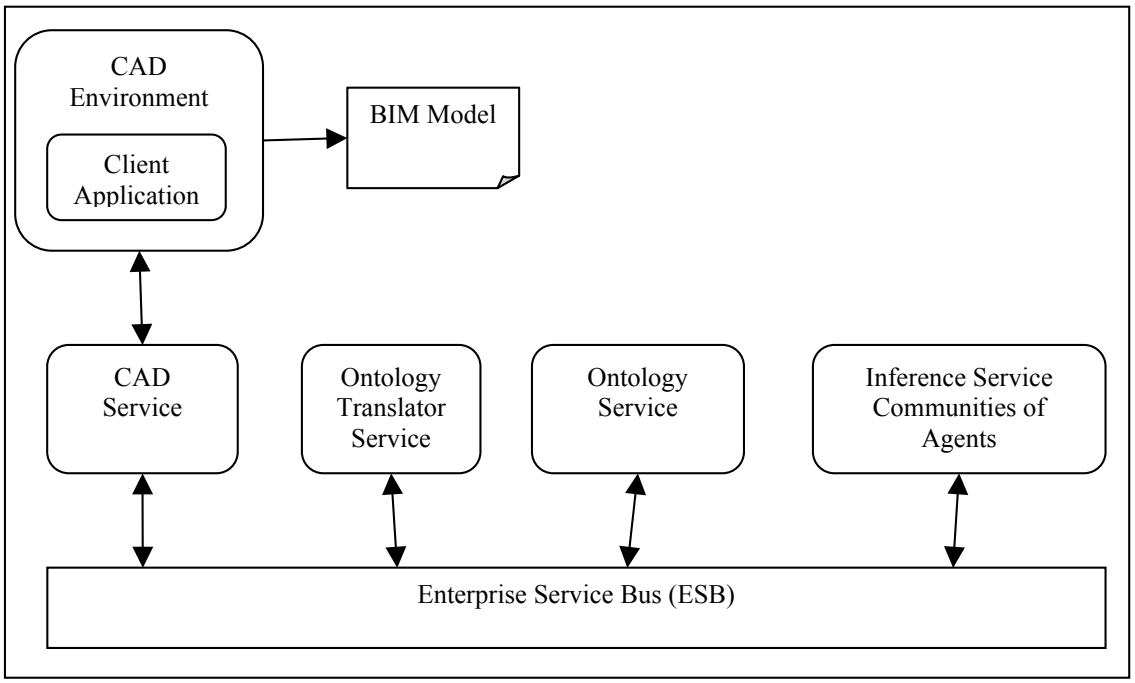


Figure 4: Diagrammatic system architecture

- A translation service that translates the BIM model into the system ontological representation to allow the higher level inferences to take place. This service is made part of the workflow through the user settings in the client application. (Taylor et. al. 2009, Pohl 2008).
- An ontology service that builds, maintains, and handles the communication of the ontology with the other services. The ontology service contains the subscription service described previously, which registers the interests of other components in ontology changes. The ontology service also builds additional relationships into the model, which was exported from BIM. The additional relationships are inferred based on the existing ones and they provide enhanced context for the inference services.
- An inference service that is made up of a number of agent communities. An agent community is a collection of agents in a given domain (e.g., energy efficiency, water use, recycling, etc.). Each agent examines the design from its perspective and produces an assessment of the quality of the design elements in that perspective. Agents may make recommendations or enhancements to the design elements and communicate the recommendations back to the ontology. The inference service is connected to the ontology service and monitors changes there, through the ontology subscription service.

6. Conclusions

Design for *sustainability* combines the complexity of traditional architectural design with the complexity of considering a host of environmental issues that are based on ecological principles, in the evolving design solution. Management of this compound complexity requires the assistance of an intelligent software system environment. There are two main requirements for such an environment. One is a rich contextual representation of design information. The second is collaboration between the human user and the software environment. The current state of technology in software development offers opportunities for developing a distributed,

collaborative, intelligent design support system. Service-oriented architecture (SOA) concepts provide the framework and the guiding principles for developing distributed, service-based systems. The field of ontological representation offers a direction for the expressive modeling of domain knowledge, which forms an enabling foundation for intelligent agents as autonomous, collaborative software tools that can monitor the evolving design, participate in problem solving in specific domains, gather and present relevant information to the designer, and communicate with the user when necessary.

References

AEDOT (1992); 'AEDOT Prototype 1.1: An Implementation of the ICADS Model'; Technical Report CADRU-07-92, Collaborative Agent Design Research Center, Cal Poly, San Luis Obispo, CA 93407.

Assal H., K. Pohl and J. Pohl (2009); 'The Representation of Context in Computer Software'; Pre-Conference Proceedings, Focus Symposium on Knowledge Management Systems, InterSymp-2009, Baden-Baden, Germany, 4 August.

Barber K., A. Goel, D. Han, J. Kim, D. Lam, T. Liu, M. MacMahon, C. Martin and R. McKay (2003); 'Infrastructure for Design, Deployment and *Experimentation* of Distributed Agent-based Systems: The Requirements'; The Technologies, and an Example, Autonomous Agents and Multi-Agent Systems. Volume 7, No. 1-2 (pp 49-69).

Blum A. and M. Furst (1997); 'Fast Planning Through Planning Graph Analysis'; Artificial Intelligence, 90 (pp.281-300).

Brown P. (2008); 'Implementing SOA: Total Architecture in Practice'; Addison-Wesley.

Dejong G. (1982); 'An Overview of the Frump System'; Lehnert and Ringle (eds.) Strategies for Natural Language Processing, Lawrence Erlbaum, Hillsdale, New Jersey (pp.149-176).

Diaz C., W. Waiters, J. Pickard, J. Naylor, S. Gollery, P. McGraw, M. Huffman, J. Fanshier, M. Parrott, S. O'Driscoll-Packer, Boone Pendergrast and Evan Sylvester (2006); 'ICODES: Technical and Operational Description'; Technical Report CDM-20-06, CDM Technologies Inc., San Luis Obispo, California, November.

Durfee E. (1988); 'Coordination of Distributed Problem Solvers'; Kluwer Academic, Boston, Massachusetts.

Durfee E. and T. Montgomery (1990); 'A Hierarchical Protocol for Coordination of Multiagent Behavior'; Proc. 8th National Conference on Artificial Intelligence, Boston, Massachusetts (pp.86-93).

Ellis C. (1989); 'Explanation in Intelligent Systems'; in Ellis (ed.) Expert Knowledge and Explanation: The Knowledge-Language Interface, Horwood, England.

Erl T. (2008); 'SOA: Principles of Service Design'; Prentice Hall.

Fu K. and T. Booth (1975); 'Grammatical Inference: Introduction and Survey'; IEEE Transactions on Systems, Man, and Cybernetics. SMC-5: (pp.95-111, 409-423).

Gero J., M. Maher and W. Zhang (1988); 'Chunking Structural Design Knowledge as Prototypes'; Working Paper, The Architectural Computing Unit, Department of Architectural and Design Science, University of Sydney, Sydney, Australia.

Hayes P. and S. Weinstein (1991); 'Construe-TIS: A System for Content-Based Indexing of a Database of News Stories'; Rappaport and Smith (eds.) Innovative Applications of Artificial Intelligence 2, AAAI Press, Menlo Park, California (pp.47-64).

ICADS (1991); ‘ICADS Working Model Version 2 and Future Directions’; Technical Report CADRU-05-91, Collaborative Agent Design Research Center, Cal Poly, San Luis Obispo, CA 93407.

Jacobs P. and L. Rau (1988); 'A Friendly Merger of Conceptual Analysis and Linguistic Processing in a Text Processing System'; Proceedings of the Fourth IEEE AI Applications Conference, IEEE Computer Society Press, Los Alamitos, California (pp.351-356).

Kibert C. (2005); ‘Sustainable Construction: Green Building Design and Delivery’; Wiley, Hoboken, New Jersey.

Michalski R. (1983); 'A Theory and Methodology of Inductive Learning'; Artificial Intelligence, Vol.20 (pp.111-161).

Mitchell T., J. Allen, P. Chalasani, J. Cheng, O. Etzioni, M. Ringuette and J. Schlimmer (1991); 'Theo: A Framework for Self-Improving Systems'; VanLehn (ed.) Architectures for Intelligence, Twenty-Second Carnegie Mellon Symposium on Cognition, Lawrence Erlbaum, Hillsdale, New Jersey (pp.323-355).

Myers L., J. Pohl, J. Cotton, J. Snyder, K. Pohl, S. Chien, S. Aly and T. Rodriguez (1993); 'Object Representation and the ICADS-Kernel Design'; Technical Report CADRU-08-93, Collaborative Agent Design Research Center, Cal Poly, San Luis Obispo, CA 93407, January.

Nibecker J., H. Larsen, X. Pan, C. Warren, R. Chambers, D. Taylor, B. Weber, J. Delos Reyes, C. Maas, and M. Porczak (2007); ‘TRANSWAY: Technical and Operational Description’; Technical Report, CDM-21-07, CDM Technologies, Inc., San Luis Obispo, CA 93401, October.

Pan J. and J. Tenenbaum (1991); ‘Toward an Intelligent Agent Framework for Enterprise Integration’; Proc. Ninth National Conference on Artificial Intelligence, vol.1, San Diego, California, July 14-19 (pp.206-212).

Pohl J. (2008); ‘Cognitive Elements of Human Decision-Making’; in Jain L. and G. Wren (eds.); Intelligent Decision Making: An AI-Based Approach; Springer Verlag, New York.

Pohl J., A. Chapman, K. Pohl, J. Primrose and A. Wozniak (1997); 'Decision-Support Systems: Notions, Prototypes, and In-Use Applications'; Technical Report, CADRU-11-97, Collaborative Agent Design Research Center, Cal Poly, San Luis Obispo, CA 93407, January.

Pohl J. and L. Myers (1994); ‘A Distributed Cooperative Model for Architectural Design’; in Carrara G. and Y. Kalay (eds.) Knowledge-Based Computer-Aided Architectural Design, Elsevier, Amsterdam, The Netherlands.

Pohl K. (2008); ‘A Translation Engine in Support of Context-Level Interoperability’; Special Issue on Ontology Driven Interoperability for Agile Applications Using Information Systems: Requirements and Applications for Agent Mediated Decision Support, Intelligent Decision Technologies, 2 (1), January.

Pohl K. (2007); ‘Enhancing the Face of Service-Oriented Capabilities’; Pre-Conference Proceedings, Focus Symposium on Representation of Context in Software, InterSymp-2007, Baden-Baden, Germany, 31 July.

Pohl K. (1996); ‘KOALA: An Object-Agent Design System’; in Pohl J. (ed.) Proc. Focus Symposium on Advances in Cooperative Environmental Design Systems, InterSymp-96, Baden-Baden, Germany, Aug.14-18 (pp.81-92), Collaborative Agent Design Research Center, Cal Poly, San Luis Obispo, CA 93407.

Schank R. and R. Osgood (1990); 'Content Theory of Memory Indexing'; Technical Report 2, The Institute for the Learning Sciences, Northwestern University.

Schank R. (1991); 'Case-Based Teaching: Four Experiences in Educational Software Design';

Sen S., M. Sekaran, and J. Hale (1994); 'Learning to Coordinate Without Sharing Information'; in National Conference on Artificial Intelligence (pp.426-431).

Simon H. (1996); 'The Sciences of the Artificial'; 3rd ed., MIT Press, Cambridge, Massachusetts (pp. 111-3).

Taylor D. and H. Assal (2009) 'Using BPM to Support Systems Interoperability'; The International C2 Journal, Vol. 3, No. 1.

Van Der Ryn S. and S. Cowan (1996); 'Ecological Design'; Island Press, Washington, DC.

Veloso M., J. Carbonell, A. Perez, D. Borrajo, E. Fink and J. Blythe (1995); 'Integrating Planning and Learning: The PRODIGY Architecture'; Journal of Theoretical and Experimental Artificial Intelligence, 7(1).

Wooldridge M., N. Jennings and D. Kinny (1999); 'A Methodology for Agent-Oriented Analysis and Design'; Proceedings Third International Conference on Autonomous Agents (Agents-99), Seattle, Washington.

Wooldridge M. (1997); 'Agent-Based Software Engineering'; IEEE Transactions on Software Engineering, 144(1), (pp.26-37), February.

On the Road to Intelligent Web Applications

Hisham Assal, Ph.D.

Collaborative Agent Design Research Center (CADRC), Cal Poly University

California Polytechnic State University (Cal Poly)

Kym J. Pohl

CDM Technologies Inc., San Luis Obispo, California, USA

Abstract

Increasing access to data sources on the Internet offers expanding opportunities for equipping intelligent applications with the content they require whether broad in scope or rich in detail. Although typically originating within the web in a *semi-structured* form, with the use of inference-based translation and analysis mechanisms such content can be transformed into useful information and ultimately into actionable knowledge. Service-Oriented Architecture (SOA) offers a platform for accessing the web as invocable resources and effectively incorporating multiple sources of data and capabilities on the Internet into enterprise applications. Adding inference capabilities to SOA-based applications not only aids in the translation of data into information thus increasing visibility into the sea of content that is the *web*, but also provides a powerful mechanism for performing the domain-centric decision making that is the heart of intelligent applications. The Web Ontology Language (OWL) offers the medium and the tools necessary to represent models of business activities as well as support native inference across related semantic concepts. In this paper the authors present an architecture for combining OWL with a SOA-based paradigm to enhance traditional web applications with powerful inference capabilities. Commensurate with a service-oriented theme, specific techniques are presented for representing the translation activity itself as a service. The paper concludes with a discussion of two distinct types of inference: one internal to the OWL model and the other externalized into intelligent agents that operate across OWL-based concepts.

Keywords: inference, SOA, OWL, intelligent analysis, web application, semantic web

Introduction

Web applications strive to take advantage of the sea of content available on the Internet. With the migration of Service Oriented Architecture (SOA) into the modern day Web, there is a growing trend towards exposing such content as web services (McKendrick 2009). Although certainly abundant in the financial, media, and search domains, standardized protocols in conjunction with painless development platforms this proliferation of web services is far-reaching into even the most informal Internet communities. In order to effectively function within this service-oriented playing field, web applications must be architected in a manner that is compatible with these service-based environments and related SOA principles (Erl 2008).

However, compatibility with a service-oriented environment is unfortunately not sufficient to effectively take advantage of the vast amounts of content available on the web. Once obtained, the limited representation of this sought-after content offers yet another hurdle. The painful reality of today's web is that the vast majority of this content is at best found in a semi-structured

form and is usually no more than sections of free text. For web applications intending to apply some reasonable degree of analysis on this content, practicality in today's data-centric Internet requires these applications to infer meaning from this otherwise context-deprived data. To support such an activity, the service-oriented fabric within which these web applications operate should be equipped with a facility capable of supporting various forms of inference. The Web Ontology Language (OWL) offers the provisions for developing semantically-enriched models where domain concepts can be represented in not only structure but also logic allowing platform-level reasoners (Bock et. al. 2008) to perform the inference activities necessary to make sense out of the sea of data that is today's Internet.

The utility of incorporating an inference capability into the platform within which web applications operate goes beyond the transformation of data into information. Whether transformed from data or originating within the context-rich Web that is the promise of the Semantic Web (Berbers-Lee et. al. 2001), once web-based content is available as *information* an entirely new set of decision-support possibilities becomes apparent. Within this environment, web applications will engage in considerably more reasoning activities than the web applications of the past. Equipping the framework within which web applications operate with the constructs and facilities that directly support such inference activities will be imperative.

Following is a discussion presenting several powerful features of OWL that can be leveraged by web applications to perform the inference necessary to transform data into information, as well as capitalize on this information to perform sophisticated analysis. The reader is then presented with a hybrid architecture that successfully integrates an OWL execution platform with a service-oriented architecture managing the bridge between these two distinct paradigms. The paper concludes with a discussion of key distinctions between a *service* and an *intelligent agent*

OWL: Web Ontology Language

The Web Ontology Language, commonly referred to simply as OWL, is a semantic markup language. The primary purpose of OWL is to facilitate the publishing and sharing of ontologies across the World Wide Web (WWW). OWL is intended to be used by software applications that need to process web-based content in a meaningful manner. In other words, OWL-based content is designed to be machine-interpretable.

A typical OWL environment consists of several key components, some to be employed at development-time and others that manage runtime activities. Together, these components form a cohesive platform for the development and execution of semantic content.

OWL Modeler

The OWL paradigm supports a number of powerful modeling concepts including dynamic, multiple classification as well as unconstrained attribute composition. As such, developing a model that takes full advantage of such features requires a development environment equipped with native and intuitive support for key modeling constructs. Further, such modeling environments should seamlessly integrate with model validation, presentation, reasoning, and code generation capabilities. There are a variety of such development tools available in off-the-shelf form including Protégé.

OWL Reasoner

Perhaps the most important component of any OWL environment is the *reasoner*. As the name implies, the main function of this component is to essentially reason about a given OWL model and its associated content. More specifically, an OWL *reasoner* processes class definitions, individuals, and rules in an effort to accomplish two primary objectives:

1. To identify any logical inconsistencies existing within the model definition and its use. Some of these inconsistencies may take the form of uninstantiable classes, conflicting definitions, and so on.
2. To identify any additional knowledge that can be automatically inferred based on the model definition and associated content. This additional knowledge can include subsumption and association relationships or the qualification of an individual for membership to under classification(s). For example, based on class definitions, one class may meet all of the criteria to be considered a subclass of another. Likewise, based upon the characteristics of a particular individual, that individual may also meet the requirements to be a member of one or more additional classifications. It should be noted that most reasoners available off-the-shelf (OTS) focus on inferring *additional* knowledge, and not necessarily managing the validity, or truth, of existing knowledge. In other words, most OTS reasoners make little attempt to retract inferred knowledge once it is no longer valid. Managing the truth of such classifications is vital in maintaining an accurate account of inferred knowledge. As such, the inability of most OTS reasoners to perform this maintenance is a serious limitation to their practical use and results in such maintenance being the responsibility of the developer.

OWL Query Engine

The ability to interrogate or ask questions of an executing OWL model is a core requirement of any knowledge-based system. In fact, this is typically the primary means by which inferencing is performed within such environments. Although certainly not a requirement, this facility is often integrated into the OWL *reasoner* itself. Such intermingling of these two capabilities makes sense since processing queries within an OWL-based paradigm often requires degrees of inferencing.

Apart from a powerful query engine, an appropriate query language must be selected and consequently supported. Such language should be powerful enough to support representing the semantic-level questions that are often posed within an OWL environment. Such questions often go beyond the classical SQL-level queries and take forms such as “is *Jennifer* a cousin of *Luke*?” or “what’s an appropriate diagnosis for these symptoms?”. Both of these examples would typically require the use of a *reasoner* in order to formulate appropriate answers.

Key Concepts Promoted by OWL

OWL supports several very powerful concepts. Although certainly not unique to the OWL paradigm, these concepts are the fundamental enablers of OWL’s support for semantic-oriented representation (i.e., models). Following is a discussion of each of these core concepts.

Multiple Classification: As the name implies, multiple classification is the ability for an entity to be classified as one or more types simultaneously. This is a very powerful capability and has significant implications on the manner in which representational models are developed. Unlike traditional, more rigid modeling paradigms where inheritance must be employed in order to extend abstract classifications, OWL modelers enjoy a very flexible environment without concern for relating classifications in order to support a single object exhibiting features defined across multiple classifications. To manage exactly which classifications are appropriate is typically the responsibility of the OWL reasoner. Comparing features exhibited by objects against requirements for class membership, the OWL reasoner can determine which classifications a particular object currently qualifies for.

Dynamic Classification: Dynamic classification is the ability of the classification of an object to change over time. Different than re-instantiating an entity under a new classification, the identity's referential integrity is preserved as its classification(s) change throughout time. This capability goes hand-in-hand with multiple classification and together these concepts create a very dynamic environment where objects can effectively mutate throughout their lifecycle. Like management of multiple classification, determining exactly what classification(s) an OWL object qualifies for at any point in time is typically the responsibility of the OWL reasoner.

Open World Assumption (OWA): Traditional database systems operate under a set of assumptions to enable the query engine to return meaningful response. These assumptions include: the closed world assumption; the unique name assumption; and, the domain closure assumption. The closed world assumption states that if a statement cannot be proven true, given the information in the database, then it must be false. The unique name assumption states that two distinct constants designate two different objects in the universe. The domain closure assumption states that there are no other objects in the universe than those designated by constants of the database.

These assumptions were reasonable in a world where a database represented all of the information available about a given domain and no external information sources were needed to perform the functions of any database application. However, with the Internet becoming a major source of information, many applications are based on access to external information from sources that may be unknown at the design stage of the application. This requires a different kind of knowledge representation, capable of dealing with the openness of the Internet. The open world assumption was adopted to allow for the relaxation of the constraints of the closed world assumption. Along with the open world assumption, the other two assumptions were also relaxed, namely, the unique name assumption and the domain closure assumption.

The open world assumption states that there can be true statements that are not contained in the current representation. The unique name assumption is dropped to allow two objects to have the same name without being considered the same object. This means that two objects are considered the same, only if there is a statement that they are. The domain closure assumption was relaxed and converted to an open domain assumption, which states that there can be other objects in the universe than those in the current representation, unless explicitly stated otherwise.

Under an open world assumption, everything is possible unless asserted otherwise. This is in stark contrast to traditional decision-support paradigms where the volume and extent of considerations is limited to what is explicitly asserted to be true about the *world* at any given time. Although operating under an open world assumption has implications on model development, it is primarily model usage and interpretation that are affected. For example, unless stated otherwise it is certainly conceivable that two otherwise distinct objects actually represent the same entity. This simple yet powerful implication can affect whether a reasoner determines an inconsistency regarding two individuals (i.e., objects) being assigned to the same end of a functional (i.e., *a multiplicity of one*) property (i.e., relationship) or inferring that these two individuals are actually the same.

Unconstrained Composition of Characteristics: Being able to assign characteristics to individuals in a manner unbounded by the blueprint of the individual's current classification(s) is a primary trigger for dynamic classification. In environments supporting this flexibility users are free to assign to and remove characteristics from objects regardless of the object's current definition or type(s). The extent of available characteristics is bound only by the range of types defined within the model together with the avoidance of any inconsistencies as prescribed by model logic. As an individual's characteristics are changed, so may the set of classification memberships the individual qualifies for. Determining exactly what changes in classification are appropriate is the responsibility of the reasoner. For example, consider that having certain characteristics, *Rusty* currently meets the qualifications to be a Person. Suppose now that Rusty is also asserted to have a tail, a feature that is not part of the class definition of *Person*. The reasoner may now determine that *Rusty* should no longer be considered a *Person*, but rather a *Dog*. This kind of type-relaxation can be a powerful means for automatically (i.e., at the framework-level) adjusting to changing conditions and is one of the most powerful features of an OWL environment.

Perhaps the most exciting aspect of environments supporting the concepts described above is the ability to off-load significant amounts of semantic processing to framework-level components (i.e., the reasoner, etc.). Activities such as managing appropriate classification which, if supported at all, were traditionally the responsibility of application-level components can now be transparently managed by the framework itself. Not only does this result in significantly less work by application developers but by internalizing such activities can lead to improved performance compared to more externalized approaches.

Supporting Architectures

It is important to realize that the concepts described above are not necessarily unique to OWL. Rather, OWL is only one environment that supports such concepts. It is certainly possible to implement concepts such as dynamic, multiple classification and unconstrained composition within traditional object-oriented environments as well. There are numerous modeling patterns and techniques that can be employed in support of such capabilities. Figure 1 provides a Unified Modeling Language (UML) model fragment illustrating how some of these concepts can be readily represented within more rigid modeling paradigms. This model fragment has two distinct sides, a knowledge side that essentially represents type information, and an operational side that represents individual entities. The model presented in Figure 1 can be read as follows.

There exist various types of things (i.e., *ThingType*). These types have varying degrees of compatibility with each other (i.e., *isDisjointWith*) as well as the types of roles (i.e., *RoleType*) that things of these types can potentially play (i.e., *canPlay*). Note that additional knowledge-level constraints can be added to the model fragment in a similar manner representing notions such as symmetry, transitivity, irreflexibility, and so on. The operational side of the model indicates that a specific thing (i.e., *Thing*) can be classified simultaneously under one or more classifications (i.e., *ThingType*). Further, depending on its characteristics a thing can change type(s) dynamically throughout time without jeopardizing its unique identity. In addition, things can play a variety of roles (i.e., *Role*) throughout time. These roles are typed according to their specific *RoleType*. The set of potential roles a thing can play is governed by the thing's current set of classifications.

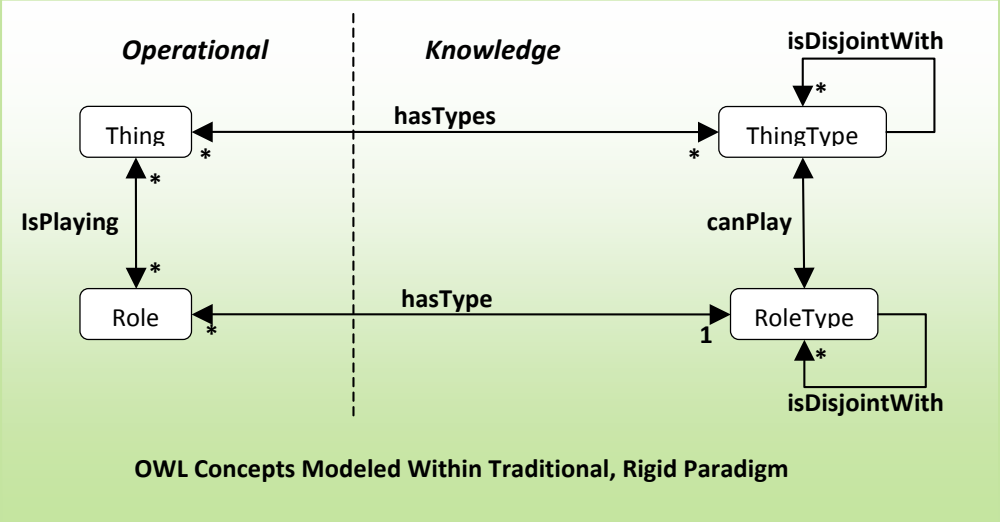


Figure 1: UML pattern supporting dynamic, multiple classification and type compatibility

However, despite the mechanics supportable within more traditional, rigid environments there is a difference between an environment where such support is *possible* and one where it is *native* to the platform. It is important that the mechanics of such support be as transparent and integrated as possible alleviating users from dealing with supporting model elements that are not directly aligned with target concepts. Although by no means unique, OWL's native support for such concepts should not go unnoticed and can translate into not only significant savings in development costs but can also result in a more elegant implementation.

However, a solely OWL-based solution may not be adequate when all factors are considered. A more hybrid approach may, in fact, offer a more balanced and successful result. The reasons for this are as follows. First, although a promising direction, there is a significant difference between the extensiveness of support and maturity of traditional enterprise environments as compared to that of OWL. Second, due in no small part to this maturity, traditional enterprise frameworks tend to be notably more refined and efficient when compared to current OWL offerings. Further, many organizations have a considerable investment in traditional toolsets and frameworks and are therefore reluctant to completely abandon such capabilities in favor of a pure OWL solution. Accordingly, it can be argued that an effective architecture should take the form of a partnership

between an OWL platform and one that comprises more traditional components. The objective of such a solution would be to promote the advantages of both environments while minimizing their limitations. The following section presents a reference architecture, which combines components that natively support OWL with those inherent in a more traditional, rigid environments.

Hybrid Architecture

The objective of the architecture proposed in this section is to combine the emerging support for OWL with the mature and extensively supported object-oriented enterprise environments currently employed by numerous organizations. The resulting architecture strives to capitalize on the benefits exhibited by each individual paradigm by segregating specific tasks to appropriately-suited mechanisms. Since this architecture exists as a sort of cross between two existing architectures, the resulting combination is referred to as a *hybrid* architecture.

The OWL Side

Considering the native manner in which the OWL platform supports the powerful concepts described earlier, it is imperative that the solution architecture include the fundamental components comprising an OWL architecture. As mentioned earlier, these components include an OWL modeler, reasoner, rule engine, and query processor. Ideally, these components would be taken from off-the-shelf offerings. However, at the time of this writing OWL is still undergoing a significant maturing process. As such, it is likely that some additional development will be necessary to elevate certain off-the-shelf components to the required level of support. It is especially imperative that the employed reasoner not only support truth maintenance but that it performs its inference on an asynchronous and continual basis. Further, the reasoner must be able to perform this monitoring across potentially large portions of model elements and, considering the frequency with which model content is likely to change in the complex, multivariable environment of decision-support operations, it must do so in as efficient a manner as possible.

The Traditional Side

The traditional side of the equation is characterized by its scalable framework directly supporting the business processes of a complex, multi-faceted organization. At the heart of the more modern-day variety are one or more well-crafted, object-oriented domain models providing the context upon which enterprise-level, decision-support activities are performed. However, although over the years enterprise application frameworks have certainly proven their worth, they do not traditionally support the level of flexibility and dynamics available in OWL.

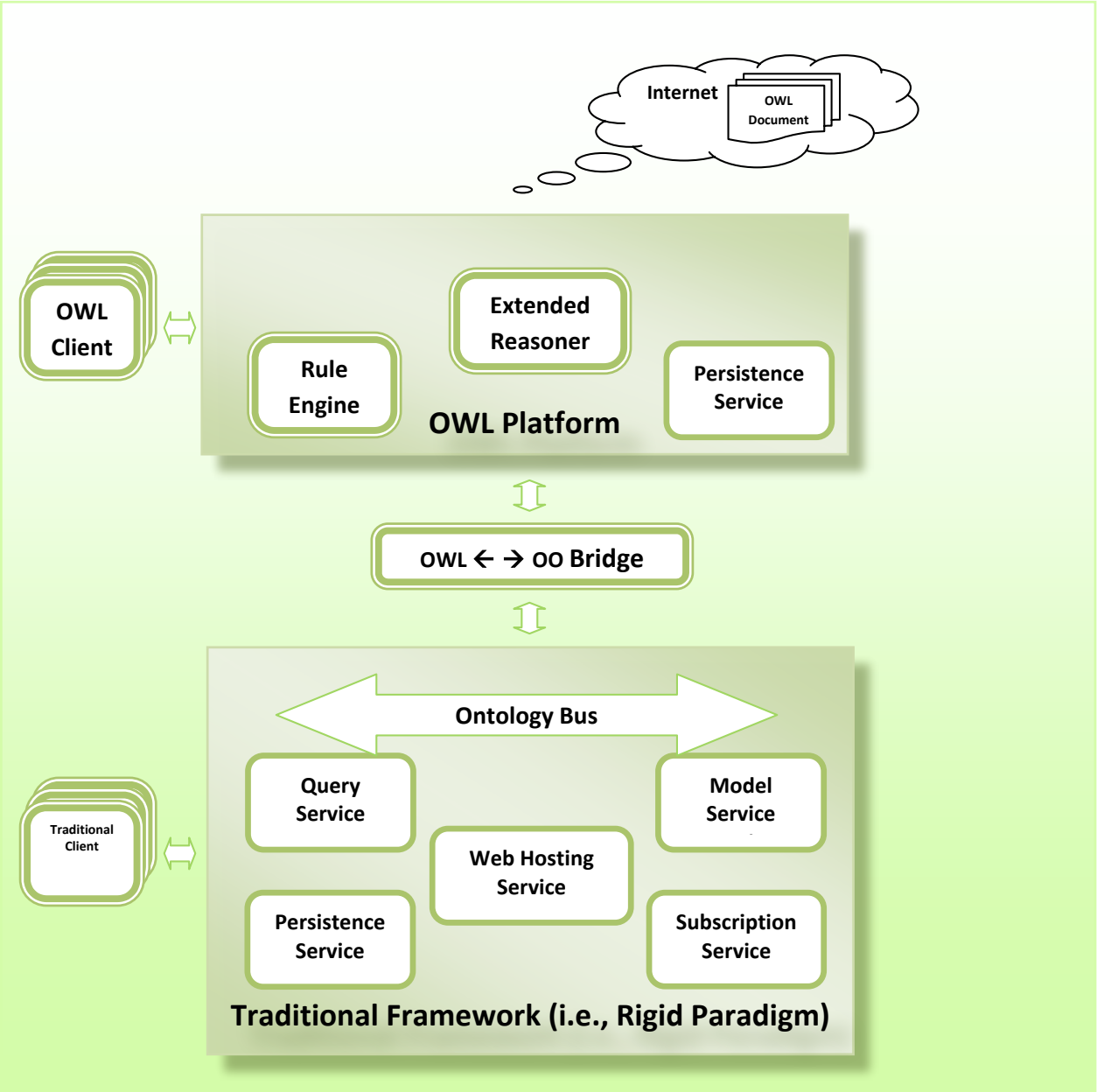


Figure 2: Hybrid architecture integrating more traditional components with those of OWL

This can be seen in the high degree of rigidity typically found with the domain models that they manage. Concepts such as dynamic classification and multiple classification, let alone unconstrained composition, are not natively supported in such paradigms and therefore are not typically engineered into the models that these environments manage. Although some of these concepts are, in fact, supportable within such rigid modeling paradigms, *explicit* representation of such concepts (Figure 1) typically adds significant complexity and overhead to already extensive domain models. Further, such environments offer limited support for the management of such concepts. Such frameworks contain limited facilities for managing the concepts that are core to OWL, including the dynamic reclassification of an entity. While such support can be developed within framework-level components, this would typically introduce notable overhead.

As a result, the exposure of elaborate model constructs to users increases the overall complexity and dilutes the otherwise domain-centric nature of the model with extraneous notions (i.e., a role, an entity-property relationship, and so on). Even if such intricate complexities were to be overlooked, the models would need to be re-architected to achieve compatibility with notions such as dynamic classification and unconstrained composition.

Considering the difficulties described above, a more realistic objective would appear to be to focus support on those concepts that directly address core use-cases found in today's decision-support systems. Although not comprehensive, this list would include the ability to support multiple views of an entity personalized to the native vocabulary, structure, and scope of individual users. To avoid the need to re-architect such users for compatibility, it may suffice to limit the scope of such multiple and dynamic classification to exist across users, and not necessarily within the scope of an individual user itself. Easing of this scope allows for users operating within platforms not directly supporting these concepts to still effectively *play* within such arenas.

The Bridge

As its name indicates, the primary purpose of the Bridge is to form a connective conduit between the two platforms, or paradigms comprising the overall architecture. Functioning much like a basic messaging service, the Bridge fulfills requests to send content from one environment to the other. To accomplish this feat, however, the Bridge must typically perform a level of translation together with occasional orchestration in order to effectively and correctly represent the content within the neighboring paradigm. Although each environment may manage its own set of native model fragments, one of the goals of this approach is to facilitate the modeling of cross-environment domain concepts within the more powerful modeling paradigm offered by OWL. With such an approach, any model fragment necessary to represent such concepts within the more rigid traditional side of the architecture would be automatically derived from the original OWL-based description. Although these shared domain concepts stem from the original OWL-based incarnation, their composition can understandably differ significantly. Within the OWL environment, such concepts are represented as a natural part of the OWL language. Whereas, in the more rigid modeling paradigm such concepts are supported through employment of specific analysis patterns (Figure 1) and consequently managed through purpose-built extensions to framework-level components (e.g., a Model Server capable of supporting multiple views, or facades).

Translating Classification

One of the core activities this hybrid architecture must support deals with passing changes in an entity's classification from one side of the architecture to the other. More specifically, the architecture must support a complex set of activities ranging from the initial determination of an entity's classification(s) to the translation and consequential mirroring of such an event within the neighboring world. As discussed earlier, the determination and consequential management of classification is a capability readily supported by any reasoner-equipped OWL platform. As such, management of an entity's classification(s) should clearly be handled by the OWL side of the equation. Therefore, it is the task of the Bridge to translate changes in an entity's classification into the model element counterparts (i.e., facades or views) offered within the traditional environment.

To help convey the key steps involved in this translation process, consider the example scenario of a weather system beginning to impede the use of a frequented section of roadway. Within the OWL environment the reasoner quickly determines that the OWL individual representing the weather system should not only be categorized under its original WeatherSystem classification but should now also qualify as a *TrafficImpediment*, for example. Reacting to this additional classification, the Bridge has the task of reflecting this event within the traditional side of the architecture ready for consumption by traditional enterprise components. As described above, within the more rigid modeling paradigm governing the traditional side of the architecture, additional classifications of an entity are represented as stateful facades, or views, overlaid upon the original entity. In this example, the original entity representing the weather system might be an instance of the *WeatherEvent* class and an impediment-oriented view of such an entity might be represented as a *RoadUsageImpediment* that derives its weather-related properties from the underlying WeatherEvent model fragment. The impediment-related properties would be the stateful part of the façade or view. Once translated into this form, users operating within the more rigid environment that are interested in seeing the weather system in its innate form would interact with it as a WeatherSystem entity. By the same token, users only interested in things that impede traffic would interact with the entity as a RoadUsageImpediment. In either case, through support for this type of OWL-like classification, users would have the ability to see and interact with entities in the form most suitable for their individualized perspectives.

This example illustrates how *hybrid* architecture capitalizes on the dynamic, multiple classification capabilities inherent within OWL in a manner compatible with somewhat more rigid, yet notably more resourced and utilized, traditional platforms. The Bridge component of the architecture provides a seamless conduit between both worlds whereby events and affects in one environment can be effectively reflected in the other.

Service-Oriented Architectures and Agent-Based Systems

The semantic web is promoted as an environment in which meaningful exchange of information can take place. Both service-oriented architectures and agent-based systems are considered paradigms that fit the semantic web concepts and work with them. Although at first glance one may think that services and agents can be used interchangeably, there are fundamental differences between the two paradigms.

Differences between Services and Agents

The advent of web technology and the desire to build distributed systems out of existing software components that may exist in different organizations gave rise to the concept of a Service-Oriented Architecture (SOA). The SOA paradigm structures a software system as a collection of services, communicating through a common facility, such as an Enterprise Service Bus (ESB). A service is a software component that performs a specific function and has a well-defined application programming interface (API). The service API defines the functions that are performed by the service and its input and output (Brown 2008).

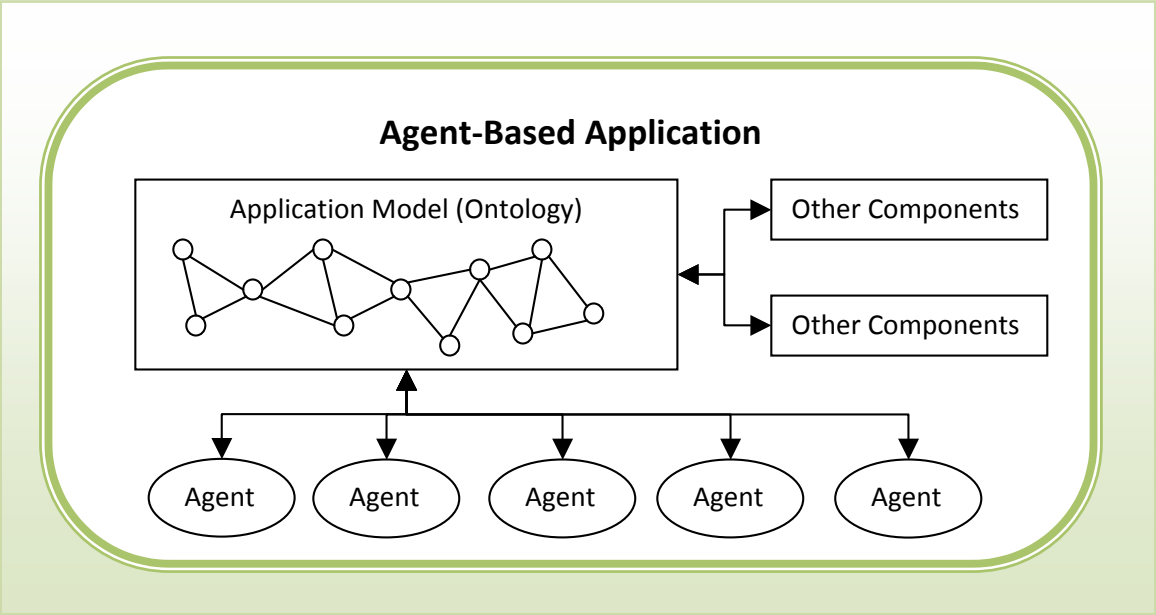


Figure 3: Typical architecture of agent-based systems

Software agents are software components that are situated in an environment and capable of flexible, autonomous action (Figure 3) (Wooldridge & Jennings 1995, Jennings et. al. 1998). Being situated in an environment means that the software component receives sensory information from the environment and can perform acts, which change the environment (e.g., create new objects, delete existing objects, change values of object attributes, or change relationships among objects). Software agents are also autonomous because they can take action without being explicitly invoked by the user. The changes in the environment trigger their action, based on their interest. Agent autonomy also means that agents have control over their internal state. They determine what information they keep internally to maintain their awareness of the environment and the current state of their interest satisfaction.

The flexibility of agents is described by three qualities: responsive; proactive; and, social. Agents are responsive because their actions are triggered in response to changes in the environment. If the information in their environment is updated and the change affects their interest, they respond to that change directly. They are proactive because they have a set of interests and they try to satisfy them by taking actions based on the available information. The social quality of agents means that agents are capable of communicating with other agents (or human users) by providing information about their current internal state, the degree of satisfaction of their interests, and possibly the reasoning behind any action they take.

There are other qualities that can be bestowed on agents, such as mobility (the ability of agents to move from one server to another and perform functions on every server they move to), learning (the ability of agents to acquire new knowledge and update their current set of interests), and intelligence (the ability of agents to perform analysis on the environment and produce recommendations, alerts, and warnings) (Wooldridge et. al. 1999; Barber et. al. 2003).

The above description of both services and agents shows that the two software components are different in fundamental ways. While agents are embedded in an environment and receive updates about the current state of that environment, services are totally unaware of any

environment external to them and have no knowledge about any system that uses them. Agents act proactively without user invocation, while services are explicitly called through a well-defined API. Agents can perform acts to change the environment, while services can only accept input and produce results that are passed onto the calling component, which decides how to use the service.

Approaches for using agents in SOA.

The two paradigms, SOA and agent-based systems, are different and serve different purposes. However, they can complement one another. One approach to combining agents with services is to build agent-based services (Figure 4). Such a service can be as simple as a unit conversion software component, or it can be as complex as a planning system with access to external databases or other data sources. In the case of complex applications, agents can be embedded within the service. They understand the internal model of the service and monitor its state. When the internal state of the service changes, agents can react and produce their analysis or cause a change in the service environment.

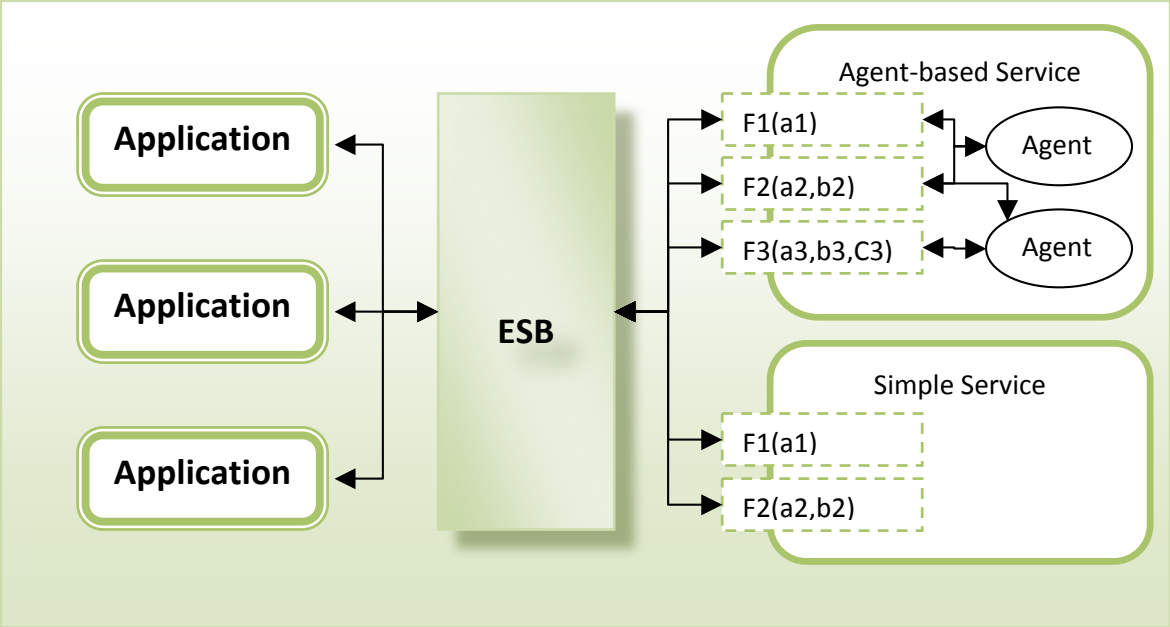


Figure 4: Agents as services

Another approach is to use agents in the main application and use services as external components (Figure 5). The client application will be a service-based application as well as an agent-based application. The application's internal model is the environment in which agents are embedded. In this architecture, agents operate on the application information, by monitoring the current state and by having the ability to take actions to change that state. The agent functions are related to the application objectives. They provide analysis that is related to the application function and may initiate requests for external services. Such requests go through the ESB like any other application request for service.

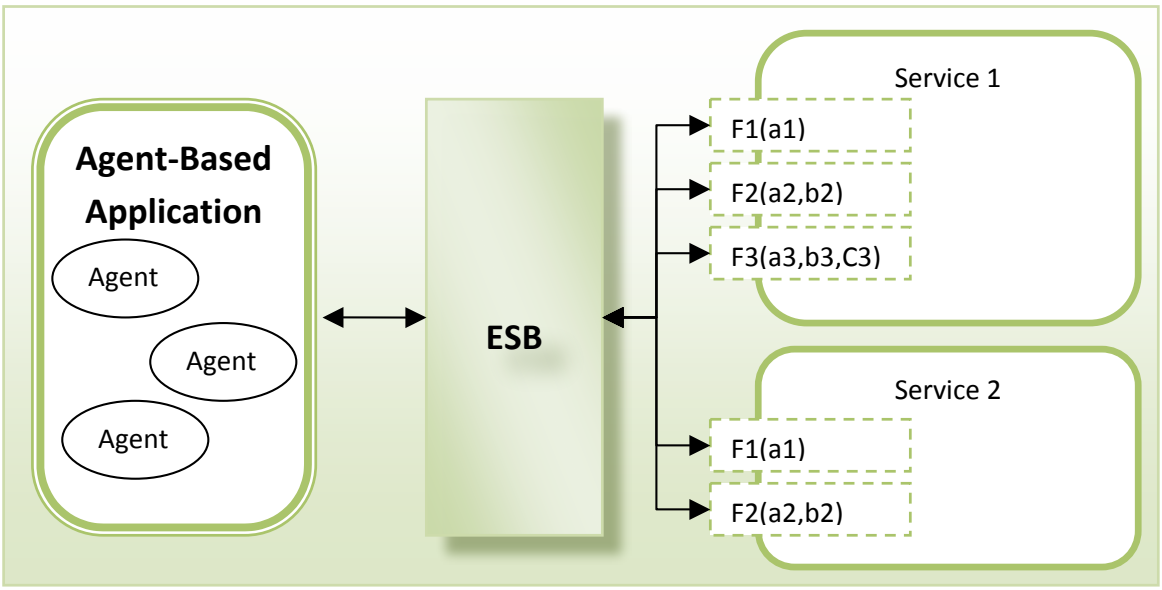


Figure 5: Agents as components that use services

Agents can also play a role within the ESB (Figure 6). Some components of the ESB provide assistance in locating, executing and monitoring services. Other components provide assistance in mediating service requests and data requirements. Intelligent agents can provide help in dealing with such issues, especially when the number of services grows and the constraints on their use and access become more complex.

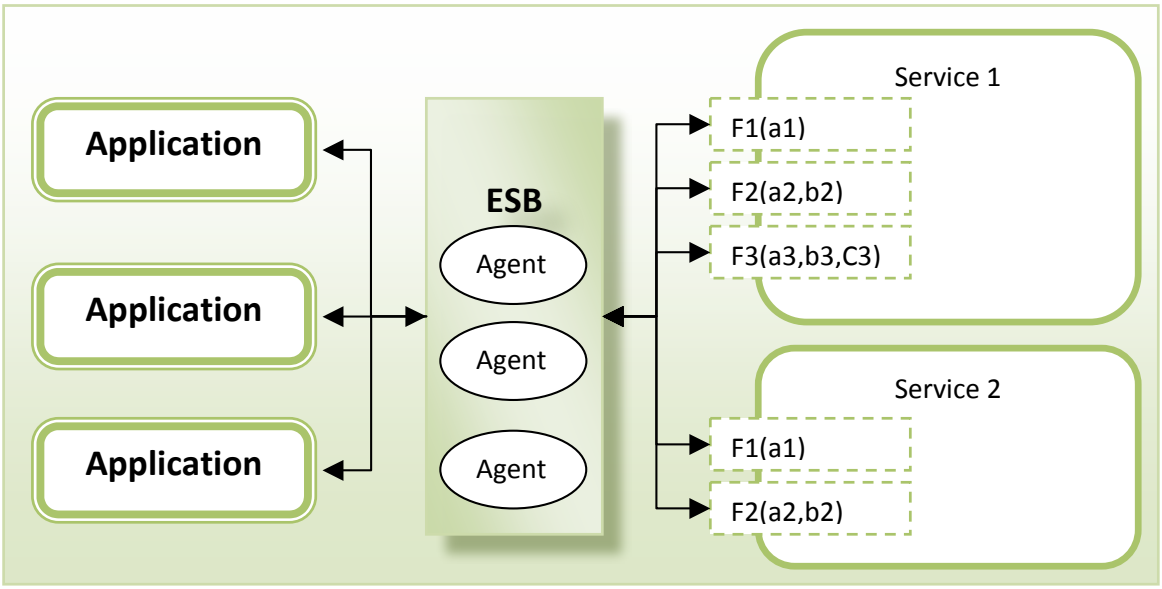


Figure 6: Service management agents

Recommendations for Using OWL to Build Systems

The Reasoner

In OWL-based software development the reasoner plays a central role in building the system. It is the component that communicates with the OWL ontology and with the other components of

the system. Therefore, the choice of a reasoner must be made carefully in order to ensure that the system components receive the information they need from the ontology in an accurate and timely manner.

It is likely that existing reasoners, such as JENA, may need to be extended to suit the specific requirements of a system. The main function of a reasoner is to determine the consistency of the current state of the ontology given a specific data set. When the data change, it is essential that the consistency check be executed again. This is time consuming and may not serve the purposes of the user. Under these circumstances it may be important to consider a truth maintenance component as an extension of the reasoner. The truth maintenance component would respond to changes in the data set and examines their implications on other associated data items. It makes the necessary adjustments to relevant data items only without examining the entire ontology data set. This capability is typically implemented through a set of rules identifying significant expected changes and specifying the appropriate responses.

It is useful for the reasoner to provide asynchronous communication with the system components, so that large updates may not affect the performance of other system components. Asynchronous communication is necessary when there are many components and each component subscribes only to a small subset of the information. Updates can be pushed to clients based on their subscription profile.

Reasoner performance may be one of the most important considerations when dealing with a large volume of data. There are reasoners that utilize the Rete algorithm (Forgy 1981), such as Bossam and FuXi. The Rete algorithm was designed to handle a large volume of data with a small number of rules. The basic idea is to build a network of patterns that represent the rule conditions and examine the data as it comes into the network for matches. This algorithm reduces the number of checks that have to be performed and makes the pattern matching more efficient.

The Database

The back-end database, which holds all the information in the OWL ontology, must be designed to support the following:

- The storage of large volumes of data.
- The storage of all the information that may be implied by the relationships among concepts.
- The ability to retrieve large chunks of data in reasonable time.

The database serves as the main repository of information from the ontology. Some level of inference can take place within the database or at the retrieval stage. SQL queries can be designed to return data that represent a given condition. The returned data are typically a small subset of the database (i.e., the ontology data) and can be further used to perform more complex processing by the querying component. In this view, SQL can be considered a preliminary inference mechanism.

Building Rules

Agents can be designed as sets of rules that fire based on the satisfaction of their conditions. Such rules can be built manually or, in some cases, automatically.

Manual Rule Development: Rules can be developed as part of the OWL ontology or as a client to another system, and receive their ontology updates through the reasoner. In both cases, the design of agent rules has to consider the following:

- Rules must fire on their own as soon as their conditions are satisfied. They should not require any user interaction to fire.
- Rules can produce additional information, which has to have representation in the ontology. This information may be alerts for the user, requests for other information, or changes to existing ontology objects.
- Agents can be represented in the ontology. This offers the opportunity to associate alerts, or other information types, with the agent that created them, and by doing that, creating the ability of tracing agent results and building justification for agent actions.
- Agent representation also allows the tracing of agent status (e.g., running, idle, has alerts, requires user attention, etc.).

Automatic Rule Development: Some rules can be generated from the current state of the ontology. For example, a set of rules to monitor the status of an organization, based on the current activities or associations of its members. The knowledge needed to create such rules may be embedded in a rule-generating agent. Such an agent monitors the objects of interest in the ontology and creates monitoring agent with the appropriate set of rules and associates it with the observed object. Dynamic rules can be generated in one of the following ways:

- Customizing a generic rule with the relevant ontology objects.
- Assembling rule components (i.e., conditions and actions) from existing representation in the ontology.
- Building rules from scratch, based on knowledge that is embedded in the rule-generating agent.

Tracing Rule Firing

The ability to trace the firing of rules and chain rule dependencies is supported in backward-chaining rule engines. In such systems, rule firing is triggered by a stated goal and the rule engine attempts to satisfy this goal by firing relevant rules. The engine keeps track of the chain of rules until the goal is achieved or it determines that the goal is unattainable. The rule chain is, typically, accessible to the user.

In forward chaining rule systems, tracing the firing of rules has to be explicitly implemented. The representation of agents in the ontology (see Manual Rule Development) can be expanded to include rules of interests (or all rules). The representation of rules includes status, associated objects, firing order, and so on. In the implementation of agents, every rule updates its representation in the ontology with relevant information. The rule tracing component utilizes this information to analyze the rule firing sequence and the associated information in any desired way and can re-construct a rule firing scenario and possibly extract explanation of agent actions.

The representation of rules should also have a rule description that provides high level explanation of the rule behavior. It is important for the user, when tracing the rule firing, to see what the rule is supposed to do. A rule description can be a simple text field associated with each rule, describing its intended use. It can also be a more complex description, generated from the structure of the rule.

Visualizing Rules

It is desirable to convey the dynamic nature of agents to the user by including some indication of the activation of agents in the user interface. Agents can have representative icons on the user screen to indicate agent status. Certain rules may also have their own graphic representation on the screen to indicate their status as well. Rule icons can be grouped into their agent icons, which can be maximized or minimized to control screen clutter and to provide better visual experience.

User interaction with agents is possible by expanding the agent graphic representation on the screen from an icon to a window, possibly with text fields to present agent messages to the user and forms to capture user input. Simple interaction may ask the user to acknowledge some alerts or turn off some warnings. More complex interaction may ask the user to guide the agent operation by providing additional information or selecting from multiple courses of actions.

A very important part of this interaction between user and agents is an explanation facility. The more intelligent the capabilities of the software the more important it is for the user to be able to ascertain why an agent has come to a particular conclusion. In the case of rule-based agents an explanation facility can include a tracking mechanism that is built directly into the rules and generates explanations automatically.

Conclusion

To develop intelligent web applications, two paradigms need to interact effectively. The Service Oriented Architecture (SOA) paradigm offers the structure and interaction management of service-oriented software components operating within a networked environment. The Web Ontology Language (OWL) offers the flexible and powerful modeling and inference capabilities necessary for software to reason over, or otherwise analyze information. Combining the two paradigms in a workable architecture offers great opportunities for developing intelligent web applications that take advantage of the distributed services capabilities as well as sources of information on the Internet. The architecture proposed in this paper provides a hybrid solution that seamlessly marries these two environments via a transactional bridge. The resulting combination supports the inference of web-based content within a service-oriented fabric that is the emerging form of the Web.

References

Barber K., A. Goel, D. Han, J. Kim, D. Lam, T. Liu, M. MacMahon, C. Martin and R. McKay (2003); 'Infrastructure for Design, Deployment and *Experimentation* of Distributed Agent-based Systems: The Requirements'; The Technologies, and an Example, Autonomous Agents and Multi-Agent Systems. Volume 7, No. 1-2 (pp 49-69).

Berners-Lee, Tim; Hendler, James; Lassila, Ora (2001). "The Semantic Web". *Scientific American*. 17 May.

Bock, Jurgen; Haase, Peter; Ji, Qiu; Volz, Raphael. (2008) Benchmarking OWL Reasoners. In AREa2008 - Workshop on Advancing Reasoning on the Web: Scalability and Commonsense; June.

Brown P. (2008); 'Implementing SOA: Total Architecture in Practice'; Addison-Wesley.

Erl T. (2008); 'SOA: Principles of Service Design'; Prentice Hall.

Forgy C. (1982); 'Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem'; Artificial Intelligence, 19 (pp. 17–37).

Jennings N., K. Sycara and M. Wooldridge (1998); 'A Roadmap of Agent Research and Development'; Autonomous Agents and Multi-Agent Systems, Vol. 1 (pp. 7-38).

McKendrick, Joe. (2009). SOA adoption trends -- what the data tells us. March 12. ZDNet News and Blogs, Service Oriented. http://www.zdnet.com/blog/service-oriented/soa-adoption-trends-what-the-data-tells-us/1679

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

Wooldridge M., N. Jennings and D. Kinny (1999); 'A Methodology for Agent-Oriented Aanalysis and Design'; Proceedings Third International Conference on Autonomous Agents (Agents-99), Seattle, Washington.

A Method to Implement Location Transparency in a Web Service Environment

Xiaoshan Pan, PhD.
xpan@cadrc.calpoly.edu
CAD Research Center, Cal Poly, San Luis Obispo, CA
CDM Technologies, Inc., San Luis Obispo, CA

Abstract

Location transparency offers some significant benefits in the areas of middleware, Service-Oriented Architecture (SOA) and Cloud Computing. However, methods for achieving location transparency in a Web service environment are scarcely presented in the literature. This paper introduces such a method by describing a design and HTTP protocol-based implementation of location transparency. A number of benefits, including support for the creation of a virtual platform and increased mobility, availability and scalability of services, are elaborated. Two significant capabilities - performance-based load balancing and failover - are demonstrated as part of the experimental results.

Key Words

Location Transparency; Web service; Service-Oriented Architecture; Cloud Computing; Virtual Platform; Intelligent Routing; Load Balancing; Failover

1. Introduction

In a Service-Oriented Architecture (SOA) environment, location transparency offers some significant benefits to service consumers, service providers and developers. When SOA is implemented using Web service technology, location transparency can be achieved through the construction of a SOA infrastructure[1] where Web services execute and interact with each other. Location transparency is an ability of a SOA infrastructure that enables service consumers and service providers to operate independently of their locations — a service consumer can consume a service without knowing where the provider is located, because the discovery of the location takes place at run-time.

From the perspective of a service consumer, location transparency creates the impression of a *virtual platform*, in which all services seem to reside within the same machine or programming space, while in reality the services may be widely distributed over a network (e.g., Internet). This also leads to the sense of a *Cloud* – "I send a request into the Cloud, and somehow it gets processed and a useful response comes back to me!" Therefore one practical usage of *virtual platform* is to enable a consumer to access remote services as though they were local (i.e., transparent access).

From the perspective of a service provider, location transparency offers advantages such as increased mobility, availability, and scalability. Location transparency enables a service consumer to break any dependency that it may have on a fixed location of a service provider.

[1] A SOA infrastructure refers to a service run-time environment that provides capabilities such as routing, location transparency, security, service mediation, and service orchestration to SOA based systems.

The service provider can be freely relocated, bringing the advantage of mobility. In turn, this allows a service provider to perform maintenance without causing service interruption by *switching on* a backup instance of the service at a different location while the service in production is taken off-line for maintenance. Additionally, when multiple providers of the same service contract exist, location transparency offers opportunities for the SOA infrastructure to perform load balancing and fault tolerance, which leads to increased service scalability. For example, when demand for a service increases, more instances can be created (e.g., through virtualization) and registered to the SOA infrastructure. When demand decreases, some service instances are taken down to free up resources for other usage.

Another benefit of location transparency is that the service location is eliminated as a concern for service consumer developers. Traditionally, the developers need the location and access details of a service which usually are specific to a service provider hosted at a physical location. With location transparency, a service provider is an abstract service contract (that can be implemented by multiple providers), and the developers are free to focus on solving business domain problems instead of making efforts to interface with (and later on be coupled with) a particular provider.

To achieve location transparency, binding the consumer with a provider must occur at run-time (instead of at design-time). More importantly, the binding needs to be dynamic—the binding should be changeable based on criteria such as the availability, performance, and service policy of service providers at any particular point in time.

2. Location Transparency in the literature

The concept of location transparency is not new. It has been explored in the area of middleware[2] research. Stal (2002) described using a proxy design pattern to achieve location transparency in a middleware:

> The basic idea behind this pattern is to introduce a proxy component as an intermediate layer between the client and the servant. The proxy resides within the address space of the client and implements exactly the same interface(s) as the servant… Using this approach, a client can remain oblivious to any details related to distribution, such as the servant location or communication protocol uses (p.72).

Fiege et al (2003) proposed to utilize publish/subscription mechanisms to achieve location transparency, which is "necessary to make existing applications mobile," and mobility is essential to the success of mobile computing, such as mobile services and devices. Belle et al (1999) described a naming and routing algorithm that could interconnect mobile entities and route messages between them, while the locations of the involved entities are transparent to each other.

The significance of location transparency also is emphasized by researchers from the SOA community. Channabasavaiah et al (2004) claimed that "SOA is an architecture with special properties, comprising components and interconnections that stress interoperability and location transparency" (p.21). Berbner et al (2005) described location transparency as "services should have their definitions and location information stored in a repository and be accessible by a

[2] Middleware is a piece of computer software that sits in-the-middle between application software, connecting software components or applications. Middleware aims to provide interoperability in support of a coherent distributed architecture and simplify complex distributed applications.

variety of clients that could locate and invoke the services irrespective of their location" (p.211). Srinivasan and Treadwell (2005) regarded location transparency as a means of conforming to one of the SOA principles – loose coupling, because it limits the coupling between services to interface agreement solely, not to some specific service implementations. Keen et al (2004) proposed an approach to use an Enterprise Service Bus (ESB) as an intermediary to "achieve location transparency by decoupling the client and service invocation" (p. 248). Brown (2008) mentioned a number of approaches to implementing location transparency in SOA, including:

- Proxy-based approach. Using this approach, "to the service user, the proxy presents what appears to be the service's interface…The proxy forwards all incoming requests to the real service interface and forwards replies from the service interface back to the service user through the proxy interface"(p.76).
- Message-based approach. This approach relies on an intermediary party – a message service broker – to facilitate communications between service consumers and service providers. "The message service interface is no longer tied to a specific destination. Instead, the message service provides a generic interface for sending and receiving messages regardless of the destination" (p.71). A service request waits in a message queue until a service provider picks it up and processes it. In so doing, the location of the service provider that processes the message is entirely transparent to the service consumer.
- Content-based approach. This approach also utilizes an intermediary party – a mediation service – to receive a service request and then forward the request to a chosen service provider. In this case, the mediation service selects a service provider for handling a request by examining the content of the request and matching it with a provider.

In the Cloud Computing paradigm, location transparency is one of the obvious features that a cloud provides. Mei et al (2008) talked about a "cloud user should not be aware of the distributed storage of data… and it is the cloud's responsibility to retrieve them for the user through location transparency" (p.468). This claim is also true when applying to the other types of resources that a cloud can provide, such as applications, platforms, and Web services. Vaquero et al (2009) listed "access transparency for the end user" as one of the primary Cloud characteristics.

However, regardless of the significance of location transparency to the areas of middleware, SOA, and Cloud Computing, how to implement location transparency in a Web service environment is scarcely presented in the literature. To date, the closest publicly-available documents on the subject are two patents, one by Loupia (2009) and the other by Chen (2009), both of which have obscured technical descriptions.

This paper presents a method for implementing location transparency as part of the capabilities of a SOA infrastructure in a Web service environment. To remain focused, the other aspects of the SOA infrastructure, such as service mediation, service security, and service orchestration, are not discussed. The rest of the paper is organized as follows: section 3 describes the design of a mechanism to achieve location transparency utilizing a Service Registry and an Intelligent Router; section 4 describes an HTTP protocol based implementation of location transparency; section 5 presents some of the experimental results; and, section 6 provides conclusions.

3. A design for location transparency

In this design, one primary component facilitating location transparency is a service registry. As far as its implementation is concerned, a service registry can be a database, a directory service, an XML file, or a UDDI[3] registry. A service registry provides a registration mechanism to service providers, enabling service consumers to discover a service provider in the registry and subsequently invoke the service provider. Figure 1 illustrates the basic idea of utilizing a service registry to facilitate location transparency. Three steps are involved: 1) a service provider is registered with a service registry; 2) a service consumer searches the service registry and discovers the service provider; and 3) the service consumer invokes the service provider. A key concept illustrated by this mechanism is that the binding between a service consumer and service provider can take place at run-time.

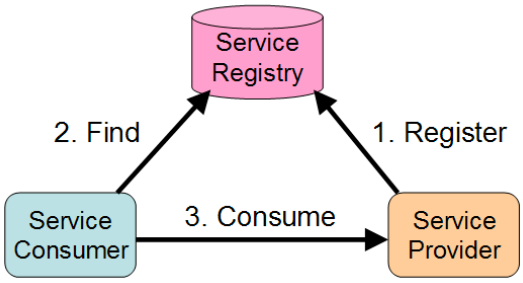


Figure 1: A service registry facilitates location transparency

Figure 1 suggests that a service consumer must perform the following steps to achieve location transparency at run-time:

1. Search a service registry for potential service providers;
2. Select a service provider if more than one is found (i.e., making routing decision); and
3. Send a request to the selected service provider and receive a response.

Assuming that steps 1 and 2 are performed by two software components, a Service Locator and a Router, respectively, we have Figure 2 below.

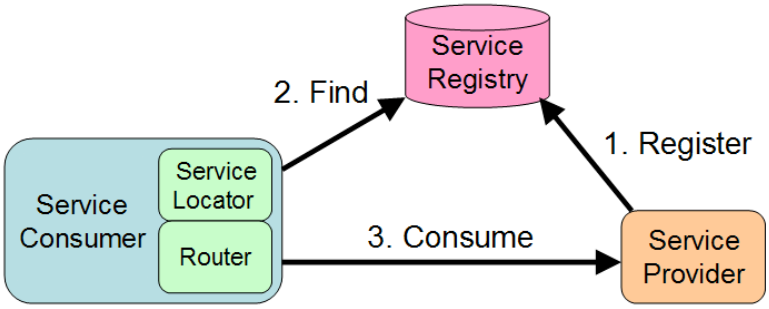


Figure 2: Service consumer embedded with a Service Locator and a Router.

Figure 2 implies that the Service Locator and the Router are part of a service consumer's internal logic, which may seem legitimate from the point of view of a single service consumer. However,

[3] UDDI refers to Universal Description, Discovery and Integration, a platform-independent, XML based registry for services to list themselves on the Internet. It enables businesses to publish service listings and discover each other and define how the services or software applications interact.

embedding these components inside a service consumer becomes problematic when multiple service consumers are involved. As illustrated in Figure 3, the Service Locator and the Router are implemented twice (i.e., Service Consumer A contains one implementation and the Service Consumer B contains the other), while the two implementations are technically identical. This introduces an implementation-redundancy issue, which is not only inefficient but also can quickly turn into a maintenance problem – just imagine hundreds of service consumers having to implement the Service Locator and the Router individually. Furthermore, from a design perspective, the focus of a service consumer is to work with business functions offered by a service provider, not finding service providers and making routing decisions.

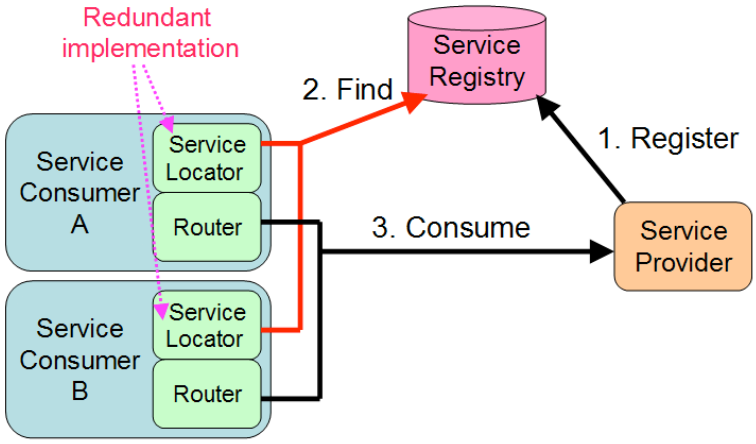


Figure 3: Redundancy implementation problem.

The SOA design disciplines advocate modularization of concerns in support of service reusability (Erl, 2008). Therefore a natural solution to the implementation redundancy problem, highlighted in Figure 3, is to make the Service Locator and the Router into separate modules that can be reused by any service consumer that would like to take advantage of location transparency. Let us call this reusable module an Intelligent Router (see Figure 4). This Router is considered intelligent because it *knows* how to locate a service provider dynamically, given a service request as its input.

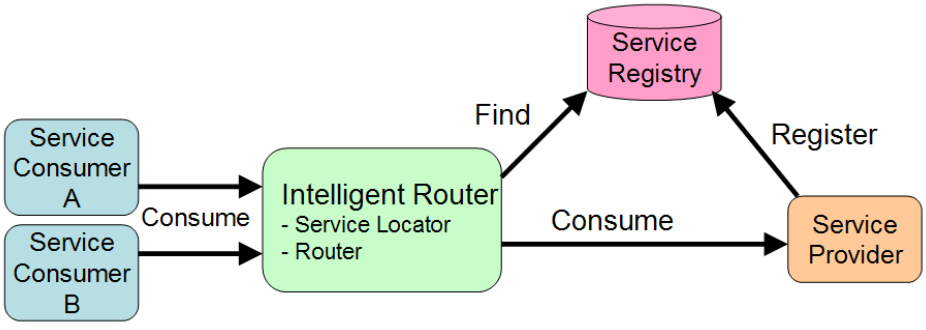


Figure 4: Utilizing an Intelligent Router to provide location transparency.

Compared to Figure 3, the design illustrated in Figure 4 simplifies the implementation of a service consumer. Furthermore, through the use of an Intelligent Router, location transparency is made available to both service consumers and service providers without them being concerned

with the implementations. Subsequently, the concept of a Virtual Platform is materialized (see Figure 5).

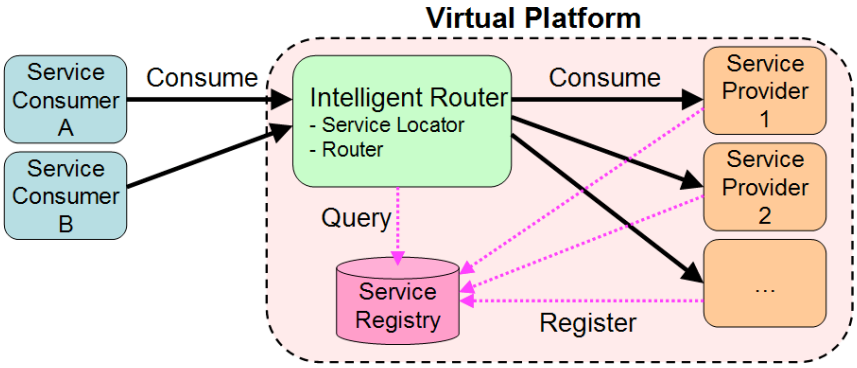


Figure 5: Creating a Virtual Platform through the use of an intelligent router and a service registry.

In Figure 5, all service providers may be dispersed throughout a network, implemented using different technologies, hosted in different environments, and removed/added to the registry at different times. However, from the perspective of a service consumer, all service providers appear as residing within the same *machine*, and all activities occurring in the *machine* are transparent to the service consumer. More importantly, when changes take place on the service providers' side, such as physical relocation of or update to a service, there is no need to make changes to the service consumers provided that the same service contracts are preserved.

4. An implementation of location transparency

The implementation described in this section assumes that Web services SOAP[4] or RESTful[5] services utilizing the HTTP protocol to transport messages, as they are currently the primary vehicles to implement SOA in the industry.

As discussed in the previous section, the core implementation of location transparency consists of two components: Service Registry and Intelligent Router. The Service Registry is well understood in the SOA community. For examples, ebXML[6] and UDDI are two industry initiatives that support the construction of a Service Registry. However, the concept of an Intelligent Router has not been fully entertained by researchers. Of the two sub-components of an Intelligent Router, the Service Locator component is relatively straightforward to construct,

[4] SOAP, or Simple Object Access Protocol, is a specification for exchanging structured information in the implementation of Web services. It relies on Extensible Markup Language (XML) as its message format, and other application layer protocols such as Remote Procedure Call (RPC) and Hypertext Transfer Protocol (HTTP) for transporting messages.

[5] REST, or Representational State Transfer, is a style of software architecture for distributed hypermedia systems. A RESTful Web service requires developers to use HTTP methods explicitly. Service contents are treated as resources that can be accessed and managed using the four basic HTTP methods – GET, POST, PUT, and DELETE.

[6] ebXML refers to Electronic Business using Extensible Markup Language and is a family of XML-based standards to provide an open, XML-based infrastructure that enables the global use of electronic business information in an interoperable, secure and consistent manner. The capabilities that it provides include publication and discovery of services electronically.

because a registry such as UDDI has a well-designed API that supports service publication and discovery. It is the Router component that poses a real challenge.

The Router must not only make intelligent routing decisions on the fly but also must act as a faithful *middle-man* between a service consumer and a service provider. From the point of view of a service consumer, the Router is a service provider, and from the point of view of a service provider, the Router is a service consumer. To perform this task, the Router must achieve *content-based routing*, meaning the content of a service request must be examined before a routing decision is made.

A traditional network router works in a very different way, which relies on a pre-defined *routing table* to perform its job, where the *routing table* is a set of fixed routing decisions, that contains lists of address mappings instructing the network router where to forward a message. In so doing, the content of a message never needs to be looked at.

With content-based routing, a router must: first, examine an incoming service request to extract information regarding *what* service contract the request applies to; second, search a service registry to discover any service providers who have implemented that service contract and *where* they are located; third, decide on a provider; fourth, create a new service request based on the original request; and finally, forward the service request to the chosen service provider. In principle, when the Intelligent Router constructs a new request from the original one, the payload of the request remains unchanged, with only the address information (i.e., addressee and return address) altered. However, there are cases where the Intelligent Router must modify the payload such as encrypting or decrypting the request or injecting security information into the message. One such example is illustrated in Figure 6.

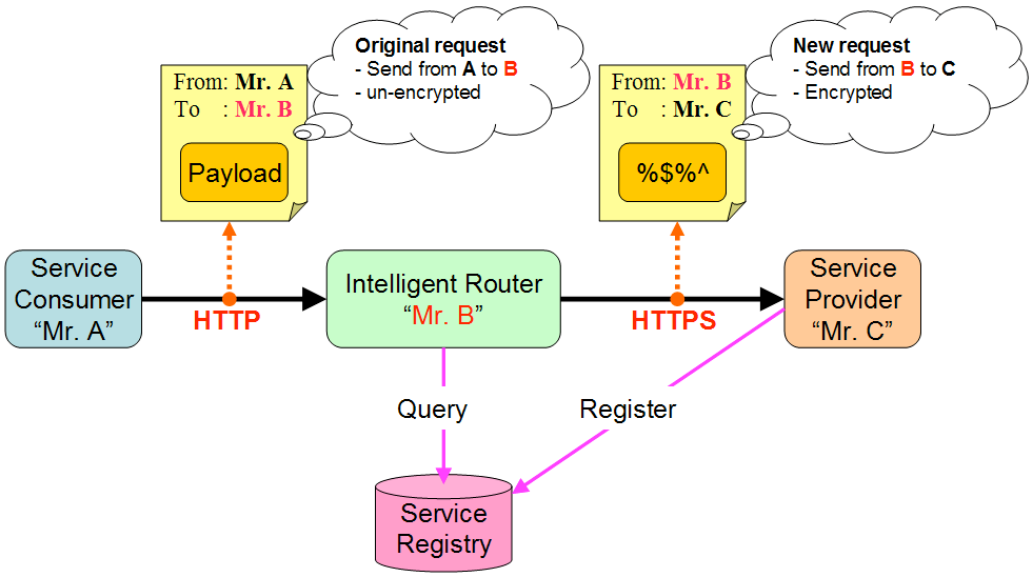


Figure 6: When performing 'content-based' routing, an Intelligent Router needs to construct a new request out of the original one.

In Figure 6, after the Intelligent Router (i.e., "Mr. B") receives a request from a Service Consumer (i.e., "Mr. A"), it makes the following modifications to the request:

- The address information of the request is changed from "From Mr. A To Mr. B" to "From Mr. B To Mr. C". "Mr. C" is the service provider chosen by the Router.

- The new request is encrypted using HTTPS, while the original request is not encrypted. This encryption step is necessary because the chosen Service Provider mandates that an incoming request be encrypted.

Similarly, when a response is received from the Service Provider, the Intelligent Router needs to construct a new response accordingly and forward it to the Service Consumer. All of the work that the Intelligent Router performs is transparent to both the Service Consumer and the Service Provider, and the Service Consumer and the Service Provider are not aware of each other's existence.

The following sections describe the implementation of a Service Registry and an Intelligent Router. The latter is composed of two sub-components: a Service Locator; and, a Router.

3.1.Service registry

The implementation discussed in this section uses OpenUDDI[7] as its service registry. OpenUDDI offers the following Application Programming Interfaces (API):

- **Publish**. This API allows a service provider to register a service with the registry so that the service can be discovered by a service consumer. In addition, this API allows a service provider to modify an existing entry in the registry.
- **Inquiry**. This API allows a service consumer to discover service providers that can satisfy its needs.

At a minimum, to publish a service instance to UDDI, a service provider must submit the following information to the UDDI registry through the Publish API: 1). Service provider's name, description and POC; 2). Service interface's name, description, contract (e.g., WSDL), and type; and, 3). Service instance's name, description, and physical end-point. An example is given as the follows.

Service provider:

Name:	Omega Cooperation
Description:	A software company that works on the Singularity technology
POC:	Dr. Omega, omega@singularity.com, Tel.: 1800.344.3444

Service interface:

Name:	Singularity Search Interface
Description:	A Web search interface into the Singularity knowledge base
Service contract:	available at https://www.singulariry-inc.com/search?WSDL
Service type:	SOAP-HTTP-Stateless

Service instance:

Name:	Singularity Search Service
Description:	A Web service that implements the Singularity Search Interface
End-point:	https://192.34.43.01:443/search-service/

Once the above information is submitted, the UDDI registry assigns a unique provider-key, interface-key, and service-key to the service provider, the service interface and the service instance, respectively. A service provider can modify the above information through the same API later on. For example, if the service provider would like to bring down the "Singularity

[7] OpenUDDI is a high performance UDDI v3 compliant service registry implementation. More information about OpenUDDI is available at: http://openuddi.sourceforge.net/

Search Service" at the location https://192.34.43.01:443 for maintenance without interrupting the consumers, the provider could do the following:

1. Activate a copy of the "Singularity Search Service" at another location, for example, https://84.32.45.03:443 – which is hosted at a different location.
2. Modify the UDDI entry such that the End-point of the service is https://84.32.45.03:443/search-service.
3. Bring down the service at https://192.34.43.01:443 and perform the maintenance.

Through the use of the Intelligent Router (introduced in the following section), the service requests previously hitting the service located at 192.34.43.01:443 would be routed to the new location at 84.32.45.03:443. Note that this location change is transparent[8] to the service consumers of the "Singularity Search Service" (see Figure 7). It is also worth noting that although the above scenario is easily achievable for stateless services more effort is required to accomplish the same for stateful services. To guarantee no service interruption to service consumers when working with a stateful service, the service consumer needs to detect a possible termination of a stateful interaction and re-send the stateful request(s) to the Intelligent Router.

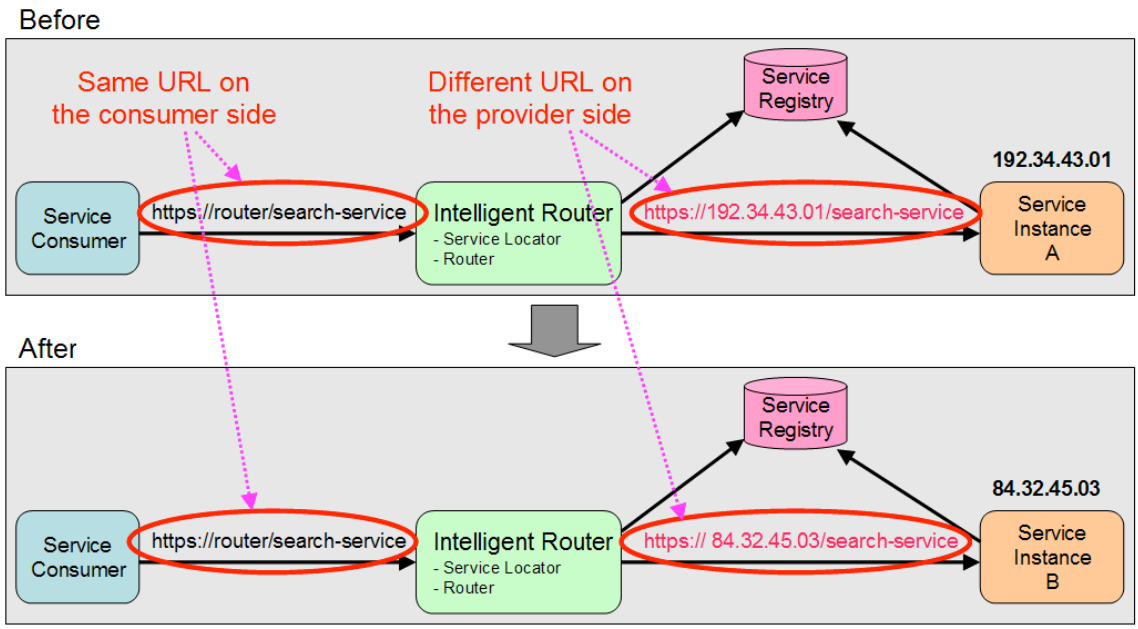


Figure 7: A service provider 'swaps' out a service instance without causing interruption to the service consumers.

3.2.Intelligent router

The Intelligent Router is composed of two sub-components: a Service Locator and a Router. Given a service request as the input, the former performs run-time queries to the OpenUDDI registry to discover service providers. The latter makes a routing decision, forwards the request to the chosen provider, and handles error conditions in the process.

[8] In order to maintain total continuity of the service, it is assumed the one of the following conditions is true: 1). the service is stateless, meaning the service does not maintain the state information of its consumers; or, 2). the service is stateful, however all state information is replicated when the copy of the service is activated.

3.2.1. Service locator

The Service Locator utilizes the UDDI Inquiry API to discover a service provider. A service request coming from a service contains a URL (Uniform Resource Locator), which is structured as follows:

[**Protocol**]://[**IP** or **DNS Name**]:[**Port**]/[**Resource URI**]

An actual example would be:

https://192.34.43.01:443/search-service/

Where "https" is the Protocol, "192.34.43.01" is the server IP, "443" is the Port, and "/search-service/" is the Resource URI (Uniform Resource Identifier).

In this implementation, a service consumer is not restricted to using Resource URI in the URL. It can send a service request to the Intelligent Router using any of the following URL formats:

1. https://router/**search-service**/ → using a URI to identify a service
2. https://router/**interface-key-23432**/ → using an interface-key to identify a service
3. https://router/**service-key-10009**/ → using a service-key to identify a service

The Service Locator will resolve #1 and #2 above to discover the service instances that match the URI "/search-service/" and the interface-key "interface-key-23432". However, #3 above will match to exactly one service instance because each service-key is uniquely assigned to a service instance in UDDI.

Assume that there are two service instances (implementing a same service contract that has the key "interface-key-23432") registered with the following end-points:

1. https://192.34.43.01:443/search-service
2. https://84.32.45.03:443/search-service

Then a service request sent to either "https://router/search-service/" or "https://router/interface-key-23432/" will cause the Service Locator to find both service instances. Another Service Locator function is to sort service instances based on their performance metrics such that a more responsive service instance would show up higher in the list. The Service Locator obtains its service metrics by sending the *testing packets*, and determining *up* or *down* status along with service responding times. A more sophisticated performance metric may be obtained if the service has a service API allowing the Service Locator to collect detailed information about the usage of CUP, heap space, physical memory, and virtual memory of the machine where the service is hosted.

3.2.2. Router

The Router performs two functions: choosing a service instance if multiple instances are found by the Service Locator; and, forwarding a service request onto a chosen service instance. If a stateful service is involved, then the Router will ensure that the service requests with the same stateful session are routed to the same service instance. The Router accomplishes this by

maintaining a cache in memory to keep track of any stateful communication between consumer and provider[9].

The following procedure describes the logic performed by the Router:

PROCEDURE: Router Logic

1. Receive a service request ***R*** from a service consumer ***C***;
2. IF ***R*** is engaged in a stateful communication with an end-point ***E***
3. THEN GOTO #14;
4. ELSE GOTO #6;
5. END IF;
6. Invoke the Service Locator and receive a list of service end-points ***L***;
7. IF ***L*** is empty
8. THEN send a 404 error response to consumer ***C***, END;
9. ELSE
10. FOR each end-point ***E*** in ***L***
11. Establish connection with ***E***
12. IF the connection fails,
13. THEN GOTO #10;
14. ELSE Construct a new request based on the original request;
15. Forward the new request to ***E***;
16. Receive a response from ***E***;
17. Construct a new response based on the original response;
18. Send the new response back to the consumer ***C***, END;
19. END IF;
20. END FOR;
21. END IF;

Although the above procedure is generic in the sense that it is applicable to most types of services in a SOA environment, the implementation of steps #14 through #18 must be protocol-specific. The following elaborations are specific to the HTTP protocol.

The general form of a HTTP request is as follows:

[HTTP Method] [URI] [Protocol/Version]
[HTTP Headers]
[Message Body]

Figure 8 depicts a sample HTTP request message.

[9] At the time of registration, a service must specify whether it is a stateful. When the Service Locator finds a service instance for a service consumer, it informs the Router if the service instance is stateful. Therefore, the Router is able to determine whether the consumer and the service instance are engaged in stateful communication.

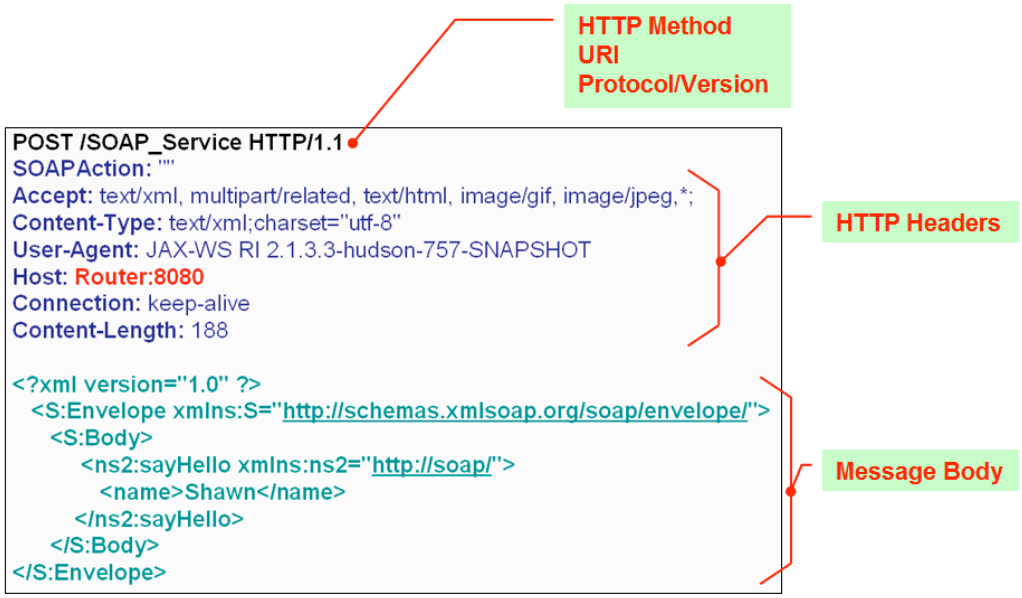


Figure 8: A sample HTTP request using SOAP.

The general form of a HTTP response is as follows:

[Protocol/Version] [Response Status]
[HTTP Headers]
[Message Body]

Figure 9 depicts a sample HTTP response message.

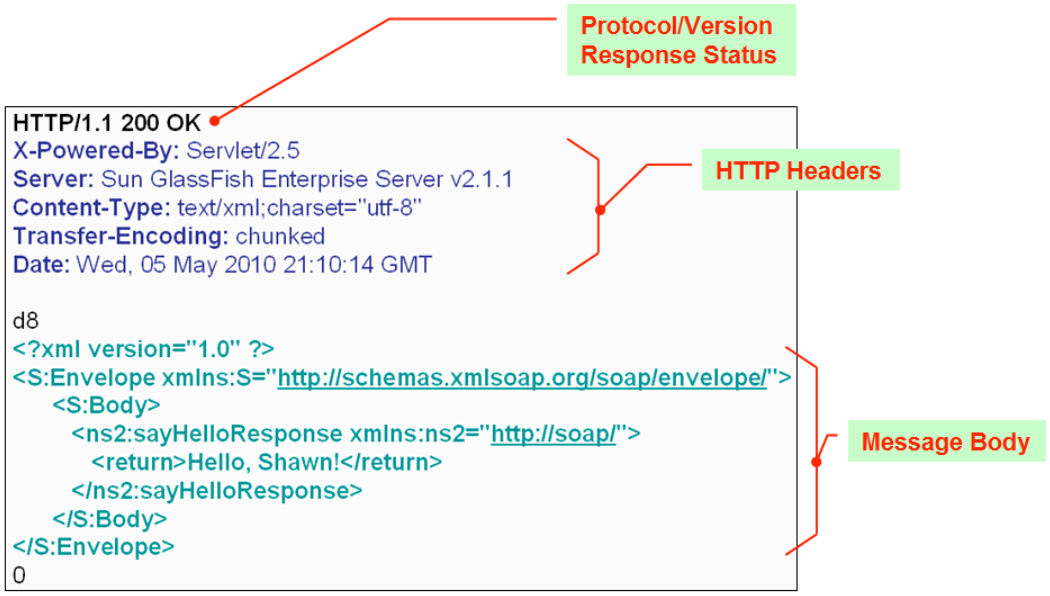


Figure 9: A sample HTTP response using SOAP

To implement step #14 and step #17 (i.e., constructing a new request and a new response), the Router needs to make changes to the *HTTP Headers* portion of a message. For example, if the service provider is hosted at "ProviderServer:9090," then the header "Host" in Figure 8 must be modified from "Router:8080" to "ProviderServer:9090," so that the correct service host is

reflected in the request. For a typical case, the HTTP headers for both a request and a response need to be modified, including:

- **Host** – specifies the Internet host and port number of the resource being requested;
- **Location** – is used to redirect the recipient to a location other than the one specified in the Request-URI;
- **Referer** – allows the client to specify the URI of the resource from which the Request-URI was obtained; and
- **Server** – is a Server response header field that contains information about the software used by the origin server to handle the request.

The implementation of step #15 and step #16 (i.e., sending a request to a provider and receiving a response) is relatively straight-forward. It requires the Router to write the service request to the *OutputStream* and read the service response from the *InputStream*, respectively, of the socket used by the Router to connect to a provider.

To implement step #11 (i.e., connecting to a provider), the Router establishes a connection with the provider using a network socket[10]. For example, using the Java language, a HTTP connection between the Router and a Provider can be created using the *java.net.Socket* class as shown in the following code sample:

```
Socket remoteServer = new Socket();
remoteServer.bind(null);
remoteServer.connect(new InetSocketAddress(IP, PORT),  TIMEOUT);
```

Where *IP* and *PORT* specify the network address of the provider, and *TIMEOUT* specifies the waiting time before a connection is terminated, in case the connection cannot be established.

For creating an HTTPS connection in Java, the *javax.net.ssl.SSLSocketFactory* class should be used to configure the Router with a proper server certificate and a certificate trust-store (to support Secure Socket Layer security),:

```
SocketFactory socketFactory = SSLSocketFactory.getDefault();
Socket remoteServer = socketFactory.createSocket();
remoteServer.bind(null);
remoteServer.connect(new InetSocketAddress(IP, PORT),  TIMEOUT);
```

Because the Router implementation described in this section does not need to examine the message body of an HTTP request or an HTTP response (other than performing encryption and decryption), the solution works generally for all HTTP-based messages (e.g., BlazeDS[11] messages).

[10] A network socket is an endpoint of a bi-directional inter-process communication flow across a computer network. Its address is identified by the combination of an IP address and a port number.

[11] BlazeDS is a server-based Java remoting and Web message technology that enables developers to easily connect to back-end distributed data and push data in real-time to Adobe Flex applications for responsive Rich Internet Application experiences.

5. Experimental results

The experiments described in this section involve using a stateless Web service and a stateful Web service as test services. The first service, *Compute_Prime_Stateless*, has one operation:

Operation: computePrime
Input: a positive integer number
Output: a list of prime numbers and the server IP

This service is stateless because it does not need to maintain any state information about the consumer of the service – the service receives an integer number and returns a list of prime numbers within the range as defined by the integer. There is no correlation between two separate service requests. In addition, the server IP that indicates the location of the server is returned for the sake of the experiment. For example, if the input is "7", then the service would return the list "2, 3, 5, 7" and "192.168.2.1", where the latter is the IP of the server that processes the request.

The second service, *Compute_Prime_Stateful*, has two operations:

Operation 1: sendInput
Input: an ID and a positive integer number
Output: none

Operation 2: compute
Input: an ID
Output: a list of prime numbers and the server IP

In order to utilize this service, a service consumer must send two consecutive requests to the service. The first request contains an ID and an integer number. After the service receives the request, it stores the ID and the number in its memory. The second request contains only an ID that the service uses to retrieve the corresponding integer in memory and to compute the prime numbers for that integer. If the ID does not exist in the memory, the service responds to the consumer with an error. This is a stateful service, because the service must keep track of the state information across two separate service requests, and the two consecutive requests must be processed by the same service instance.

Figure 10 illustrates the configuration of the experimental environment. A Service Registry and an Intelligent Router are deployed to a server named *Router*. Four service providers are registered with the Service Registry. Each service provider has a unique IP address and hosts a *Compute_Prime_Stateless* service and a *Compute_Prime_Stateful* service. A service client sends service requests to the *Router* server only. The *Router* server is responsible for locating service providers to fulfill a service request. This configuration represents a *Virtual Platform*, because from the perspective of the service consumer all service providers reside on the *Router* server.

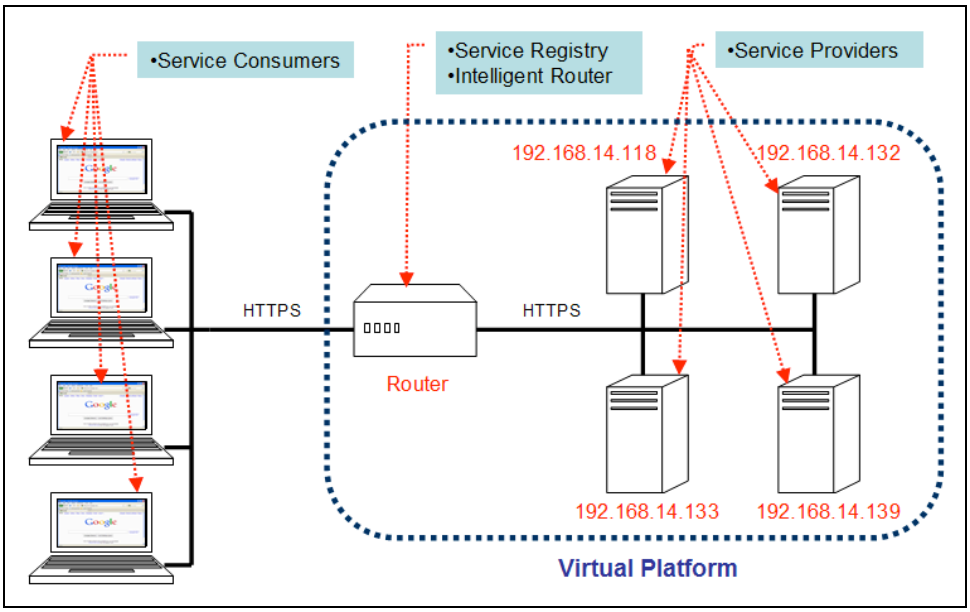


Figure 10: Experimental environment setup.

For the first experiment, four service consumer machines were configured to invoke the *Compute_Prime_Stateless* service concurrently. Each consumer machine sent out 1,000 consecutive requests (4,000 requests total), and each request caused a *Compute_Prime_Stateless* service to compute and return prime numbers between 1 and 100,000, along with the IP of the server that performed the computation. All consumers sent their requests to the following end-point (where the Intelligent Router resides):

https://Router:443/Compute_Prime/Compute_Prime_StatelessService

The *Router* server received the requests and performed load-balancing – distributing the requests to the four service providers based on their run-time performance scores. Table 1 shows the distributions of the requests across the four providers.

Table 1: Distribution of 4,000 stateless service requests across four providers

	Consumer 1	Consumer 2	Consumer 3	Consumer 4	Total
Provider 192.168.14.118	104	107	106	105	422
Provider 192.168.14.132	216	216	217	216	865
Provider 192.168.14.133	360	356	349	354	1287
Provider 192.168.14.139	320	321	328	325	1294

Similarly, the *Computer_Prime_Stateful* service was used for the second experiment. Each of the four consumer machines sent out 1,000 pairs of requests to the Router machine at the following end-point:

https://Router:443/Compute_Prime/Compute_Prime_StatefulsService

Each pair of requests consists of two consecutive requests that share the same HTTP session ID, which allows the Router to deliver the two requests to the same provider. In so doing, stateful interactions between consumers and providers are maintained. Table 2 shows the distributions of 4,000 pairs of stateful requests across the four providers.

Table 2: Distribution of 8,000 (i.e., 4,000 pairs) stateful service requests across four providers

	Consumer 1	Consumer 2	Consumer 3	Consumer 4	Total
Provider 192.168.14.118	143	142	141	142	568
Provider 192.168.14.132	231	233	230	232	926
Provider 192.168.14.133	323	321	323	322	1494
Provider 192.168.14.139	303	304	306	304	1217

The data shown in the Table 1 and the Table 2 leads to the following observations:

- Location transparency has been achieved for both the stateful and stateless services in the experiments. As far as a service consumer is concerned, there was only one provider and it resided on the server named *Router*. However, in the experiment there were multiple providers, and each was hosted on a different server.
- A performance-based load balancing capability has been achieved in these experiments. The provider with IP *192.168.14.133* has the best run-time performance, and the provider with IP *192.168.14.118* has the worst run-time performance.
- Location transparency is a suitable strategy for making a service scalable. If the demand of a service increases, more provider machines that host the service can be stood up to meet the demands. To make an additional service instance available to the consumers, the only configuration required is to register the service instance with the Service Registry.

Another significant feature supported by location transparency is failover. Specifically at runtime when multiple providers are available to support the same service contract, if one provider fails to process a request, the subsequent requests can be routed to other providers. Moreover, if a service consumer is configured to resend a stateless service request, or all requests involved in a stateful session, when a server error is detected while processing the request, then subsequent requests along with any failed requests can be recovered. In this way, it is possible to swap service providers at runtime without causing service interruptions.

In the next experiment, using the same environment illustrated in Figure 10, two service consumer machines were configured to send stateless requests (1,000 consecutive requests for each consumer) and the other two service consumer machines were configured to send stateful request pairs (1,000 pairs for each consumer) to the Router machine for processing. Each stateless request or stateful request pair will cause a service provider to compute all prime numbers between 1 and 100,000. In addition, the service consumers were configured to resend a stateless request or stateful request pair if a server error was detected. To simulate server error conditions, every 30 seconds a service provider was randomly chosen to disconnect from the network and reconnect back to the network 10 seconds later. Table 3 lists the distribution of both stateless and stateful requests that were successfully processed even though all the service providers failed to respond occasionally. As the results indicate, no single request failed to be processed even when error conditions took place.

Table 3: Distribution of both stateless and stateful requests that were successfully processed when service providers failed to respond occasionally

	Stateless Consumer 1	Stateless Consumer 2	Stateful Consumer 1	Stateful Consumer 2	Total
Provider 192.168.14.118	200	205	239	239	883
Provider 192.168.14.132	451	458	384	390	1683
Provider 192.168.14.133	115	106	140	148	2566
Provider 192.168.14.139	234	231	237	223	925
Requests re-sent	10	12	12	14	48

The data shown in the Table 3 demonstrates that a robust failover capability can be developed based on location transparency. When the failover capability works together with the load balancing capability, improved service availability can be achieved in a potentially unreliable computing environment, characterized by fluctuating network connectivity and occasional server failures.

6. Conclusions

Although the significance of location transparency is recognized in the areas of middleware, SOA, and Cloud Computing research, methods for achieving location transparency in a Web service environment are scarce. This paper presents such a method by describing a design and HTTP protocol-based implementation of location transparency in a Web service environment. In the design, the utilization of a service registry and an intelligent router is elaborated. An HTTP protocol-based implementation is presented and some experimental results are discussed. The benefits of location transparency demonstrated, include: 1) support for the creation of virtual platforms; 2) increased mobility, availability and scalability for service providers; and, 3) the elimination of service location as a concern for service consumers. In addition, two significant capabilities are established through the use of location transparency and are demonstrated, namely: performance-based load balancing; and, failover.

References:

Brown, P., Implementing SOA: Total Architecture in Practice. Addison-Wesley: Boston. 2008. ISBN 0321504720.

Berbner, R., Grollius, T., Repp, N., Heckmann, O., Ortner, E., Steinmetz, R., "An Approach for the Management of Service-Oriented Architecture (SOA) Based Application Systems. Proceedings of the Workshop Enterprise Modeling and Information Systems Architectures (EMISA 2005). October 2005, 208–221.

Belle, W., Verelst, K., and D'Hondt, T., "Location Transparent Routing in Mobile Agent Systems—Merging Name Lookups with Routing," Proc. 7th IEEE Workshop Future Trends of Distributed Computing Systems (FTDCS 99), IEEE CS Press, Los Alamitos, Calif., 1999, pp. 207-212.

Chen, J., "Reroute of a Web Service in a Web Based Application," Patent, Greenblum & Bernstein PLC, 2009. Available at: http://www.faqs.org/patents/app/20090094314

Channabasavaiah, K., Holley, K., and Tuggle, E., "Migrating to a service-oriented architecture," white paper, IBM, April 2004.

Erl, T., SOA: Principles of Service Design. Prentice Hall: New York. 2008. ISBN0132344823.

Fiege, L., Gartner, C., Kasten, O., and Zeidler, A., "Supporting Mobility in Content-Based Publish/Subscribe Middleware," in Proceedings of the ACM/IFIP/USENIX International Middleware Conference (Middleware 2003). Rio de Janeiro, Brazil, 2003, pp. 103-122.

Keen, M., Acharya, A., Bishop, S., Hopkins, A., Milinski, S., Nott, C., Robinson, R., Adams, J., and Verschueren, P., "Patterns: Implementing an SOA using an Enterprise Service Bus". IBM Redbook, July 2004.

Liupia, D., "Method of Redirecting Client Requests to Web Services," Patent, Hoffman Warnick LLC, 2009. Available at: http://www.faqs.org/patents/app/20090019106

Mei, L., Chan, W., and Tse, T., "A Tale of Clouds: Paradigm Comparisons and Some Thoughts on Research Issues," Asia-Pacific Services Computing Conference (APSCC '08), Yilan, Taiwan, December 2008, 464-469.

Srinivasan, L., and Treadwell, J., "An overview of service-oriented architecture, web services and grid computing," November 2005. Available at: http://devresource.hp.com/drc/technical papers/grid soa/SOA-Grid-HP.pdf.

Stal, M., "Web Services: Beyond Component-based Computing". Communications of the ACM, October 2002. Vol. 45, No. 10, pp. 71-76.

Vaquero, L., Rodero-Marino, L., Caceres, J., Lindner, M., "A break in the clouds: towards a cloud definition," SIGCOMM Computer Communication Review, 39 (2009), 137–150.

A Multilingual Algorithm of Texts' Semantic-Syntactic Analysis for Adaptive Planning Systems

Vladimir A. Fomichov

Department of Innovations and Business
in the Sphere of Informational Technologies
Faculty of Business Informatics
State University – Higher School of Economics
Kirpichnaya str. 33, 105679 Moscow, Russia

vdrfom@aha.ru and vfomichov@hse.ru

Abstract

The natural language texts (NL-texts) from the newspapers, e-mail lists, various blogs, etc. are the important sources of information being able to stimulate the elaboration of a new plan of actions. The paper describes a new formal approach to developing multilingual algorithms of semantic-syntactic analysis of NL-texts. It is a part of the theory of K-representations - a new theory of designing semantic-syntactic analyzers of NL-texts with the broad use of formal means for representing input, intermediary, and output data. The current version of the theory is set forth in a monograph published by Springer in 2010. One of the principal constituents of this theory is a complex, strongly structured algorithm SemSynt1 carrying out semantic-syntactic analysis of texts from some practically interesting sublanguages of the English, German, and Russian languages. An important feature of this algorithm is that it doesn't construct any syntactic representation of the inputted NL-text but directly finds semantic relations between text units. The other distinguished feature is that the algorithm is completely described with the help of formal means, that is why it is problem independent and doesn't depend on a programming system. The peculiarities and some central procedures of the algorithm SemSynt1 are analyzed.

Keywords

Semantics-oriented natural language processing; semantic representation; theory of K-representations; formal model of a linguistic database; SK-languages; multilingual algorithm of semantic-syntactic analysis

Introduction

An important source of information being able to stimulate the elaboration of a new plan of actions are the natural-language texts (NL-texts) from newspapers, e-mail lists, various blogs, etc. There are numerous situations when the information being able to change a plan of actions can be obtained from the sources in several natural languages. For instance, it is the case of planning the delivery of the loads across different countries with several languages. It would be very expensive to develop for each concrete language of possible interest a separate conceptual information retrieval system with the ability of understanding just this particular language. That

is why during last years many researchers have indicated the necessity of designing multilingual algorithms of semantic-syntactic analysis of NL-texts (see, e.g., (Wilks and Brewster 2006)).

In the monograph (Fomichov 2010) a new theory of designing multilingual semantic-syntactic analyzers of NL-texts with the use of formal means for representing input, intermediary, and output data is proposed. This theory is called the theory of K-representations (knowledge representations). Let's consider its structure.

The *first basic constituent* of the theory of K-representations is the theory of SK-languages (standard knowledge languages). The kernel of the theory of SK-languages is a mathematical model describing a system of such 10 partial operations on structured meanings (SMs) of natural language texts (NL-texts) that, using primitive conceptual items as "blocks", we are able to build SMs of arbitrary NL-texts (including articles, textbooks, etc.) and arbitrary pieces of knowledge about the world. The analysis of the scientific literature on artificial intelligence theory, mathematical and computational linguistics shows that today the class of SK-languages opens the broadest prospects for building semantic representations (SRs) of NL-texts (i.e., for representing meanings of NL-texts in a formal way).

The expressions of SK-languages will be called the K-strings. If *Expr* is an expression in natural language (NL) and a K-string *Semrepr* can be interpreted as a semantic representation of *Expr*, then *Semrepr* will be called a K-representation (KR) of the expression *Expr*.

The *second basic constituent* of the theory of K-representations is a broadly applicable mathematical model of a linguistic database. The model describes the frames expressing the necessary conditions of the existence of semantic relations, in particular, in the word combinations of the following kinds: "Verbal form (verb, participle, gerund) + Preposition + Noun", "Verbal form + Noun", "Noun1 + Preposition + Noun2", "Noun1+ Noun2", "Number designation + Noun", "Attribute + Noun", "Interrogative word + Verb".

The *third basic constituent* of the theory of K-representations is a complex, strongly structured algorithm carrying out semantic-syntactic analysis of texts from some practically interesting sublanguages of English, Russian, and German languages. The algorithm *SemSynt1* transforms a NL-text in its semantic representation being a K-representation (Fomichov 2010). The input texts can be from the English, German, and Russian languages. That is why the algorithm *SemSynt1* is multilingual.

An important feature of this algorithm is that it doesn't construct any syntactic representation of the inputted NL-text but directly finds semantic relations between text units. The other distinguished feature is that a complicated algorithm is completely described with the help of formal means, that is why it is problem independent and doesn't depend on a programming system. The algorithm is implemented in the programming languages PYTHON and C++.

The principal goals of this paper are as follows: (a) to attract the attention of the researchers to a new method of developing multilingual algorithms of semantic-syntactic analysis of texts (an implementation of this method is described in Chapters 7 – 10 of the monograph (Fomichov 2010)); (b) to illustrate the peculiarities of the central procedure of the algorithm *SemSynt1,*

allowing for the discovery of possible semantic relations in the combinations “Verbal form + Preposition (possibly, empty) + Noun Group”; (c) to explicitly add the parameter *language* to the input data of the algorithm *SemSynt1* and to add the attributes with the index *language* to the attributes of several semantic-syntactic dictionaries (relations) being the parts of the considered relational linguistic database.

Morphological and Classifying Representations of an Input Text

Morphological representation. Skipping mathematical details, we'll suppose that a morphological representation (MR) of a text T with the length *nt* is a two-dimensional array *Rm* with the names of columns *base* and *morph* (more exactly, *morph* is the designation of a group of colums), where the elements of the array rows are interpreted in the following way. Let *nmr* be the number of the rows in the array *Rm* that was constructed for the text T, and *k* be the number of a row from the array Rm, i.e. $1 \le k \le nmr$. Then *Rm[k, base]* is the basic lexical unit (the lexeme) corresponding to the word in the position p from the text T. Under the same assumptions, *Rm[k, morph]* is a sequence of the collections of the values of morphological characteristics (or features) corresponding to the word in the position *p*.

Example. Let T1 be the question "Has the management board of the firm “Rainbow” changed in May?", and T1germ be the same question in German “Hat der Verwaltungsrat der Firma “Rainbow” in Mai veraendernt sich?”. Then the morphological representation Rm1 of T1 consists of the rows *(change, md[1]), (management-board, md[2]), (of, md[3]), (firm, md[4]), (in, md[5]), (May, md[6])*, where *md[1], ..., md[6]* are the sequences of the values of morphological properties associated with the corresponding lexical units from T1. Similarly, the morphological representation Rm2 of T1germ consists of the rows *(sich-veraendern, mdg[1]), (Verwaltungsrat, mdg[2]), (Firma, mdg[3]), (in, mdg[4]), (Mai, mdg[5])*, where *mdg[1], ..., mdg[5s]* are the sequences of the values of morphological properties associated with the corresponding lexical units from T1germ.

Classifying representation. From informal point of view, we will say that a classifying representation (CR) of the text T coordinated with the morphological representation *Rm* of the text T is a two-dimensional array *Rc* with the number of the rows *nt* and the column with the indices *unit, tclass, subclass, mcoord*, in which its elements are interpreted in the following way. Let *k* be the number of any row in the array *Rc* i.e. $1 \le k \le nt$. Then *Rc[k, unit]* is one of elementary meaningful units of the text T, i.e. if $T = t_1 \dots t_{nt}$, then such position *p*, where $1 \le p \le nt$, can be found that $Rc[k, unit] = t_p$. If *Rc[k, unit]* is a word, *then Rc[k, tclass], Rc[k, subclass], Rc[k, mcoord]* are correspondingly a part of speech, a subclass of the part of speech, the sequences of the values of morphological properties. If *Rc[k, unit]* is a construct (i.e. a value of a numeric parameter), then *Rc[k, tclass]* is the string *constr*, *Rc[k, subclass]* is the designation of the subclass of informational units corresponding to this construct, *Rc[k, mcoord] = 0*.

Example. Let T1 = "Has the management board of the firm “Rainbow” changed in May?". Then a classifying representation *Rc1* of the text T1 coordinated with the morphological representation *Rm1* of T1 may be the following array:

unit	tclass	Subclass	mcoord
has-changed	verb	verb-in-indic-mood	1
the management-board	noun	common-noun	2
of	prep	nil	3
the-firm	noun	common-noun	4
“Rainbow”	artif-name	nil	0
in	prep	nil	5
May	noun	proper-noun	6
?	marker	nil	0

If T1germ =“Hat der Verwaltungsrat der Firma “Rainbow” in Mai veraendernt sich?”, then a classifying representation *Rc2* of the text T1germ coordinated with the MR *Rm2* of T1 may have the following form:

unit	tclass	subclass	mcoord
hat-veraendernt-sich	verb	verb-in-indic-mood	1
den-Verwaltungsrat	noun	common-noun	2
der-Firma	noun	common-noun	3
“Rainbow”	artif-name	nil	0
in	prep	nil	4
Mai	noun	proper-noun	5
?	marker	nil	0

The Projections of the Components of a Linguistic Basis on the Input Text

Let *Lingb* be a linguistic basis (see Chapter 7 of (Fomichov 2010)), and *Dic* be one of the following components of *Lingb*: the lexico-semantic dictionary *Lsdic*, the dictionary of verbal-prepositional semantic-syntactic frames *Vfr*, the dictionary of prepositional semantic-syntactic frames *Frp* (see Chapter 8 of (Fomichov 2010)). Then the projection of the dictionary *Dic* on the input text T is a two-dimensional array whose rows represent all data from *Dic* linked with the lexical units from T .

Let's introduce the following denotations: *Arls* is the projection of the lexico-semantic dictionary
Lsdic on the input text T; *Arvfr* is the projection of the dictionary of verbal-prepositional frames Vfr on the input text T ; *Arfrp* is the projection of the dictionary of prepositional frames *Frp* on the input text T.

Example. Let T1 = "Has the management board of the firm “Rainbow” changed in May?". Then the projection of the lexico-semantic dictionary *Lsdic* on the input text T1 may be the following two-dimensional array:

ord	sem	st1	st2	st3	comment
1	change1	event	nil	nil	Yves has changed 700 franks
1	change2	event	nil	nil	The city has changed very much in the 1990s - 2000s
2	manag-board	org	ints	phys.ob	Management board of a company
4	Company1	org	ints	phys.ob	The firm IBM
5	“Rainbow”	artif-name	nil	nil	nil
7	May	month-value	nil	nil	nil

Here the elements of the column ord are the numbers of the corresponding rows of the classifying representation *Rc1*; the sorts *org, ints, phys.ob* are interpreted as the designations of the notions “an organization”, “an intelligent system”, and “a physical object”. The sorts *ints* and *phys.ob* characterize from different standpoints the elements (people) of any firms and management boards of the firms.

The verb “to change” has more than two meanings. That is why for real computer applications this array will be a subarray of the projection of the lexico-semantic dictionary *Lsdic* on the input text T1.

Example. If T1 = "Has the management board of the firm “Rainbow” changed in May?", the projection of the dictionary of verbal-prepositional semantic-syntactic frames *Vfr* on the input text T1 *Arvfr1*:may include the following subarray *Arvfr1fragm*:

nb	semsit	lang	fm	refl	vc	trole	sprep	grc	str	expl
1	change1	eng	indic	nrf	actv	Money-sum	nil	1	money-value	ex1
1	change1	eng	indic	nrf	actv	Location	nil	1	space-ob	ex2
1	change1	eng	indic	nrf	actv	Time	on	0	moment	ex3
1	change2	eng	indic	nrf	actv	Focus-object	nil	0	phys.ob	ex4
1	change2	eng	indic	nrf	actv	Start-time	since	0	moment	ex5
1	change2	eng	indic	nrf	actv	Time-interval	during	0	moment	ex6

Here the elements *eng, indic, nrf, actv* are interpreted as the values *English, indicative-mood, non-reflexive, active-voice* of the properties *language, form-of-verb, reflexivity, voice*; the

elements *Money-sum, Location, Time, Focus-object, Start-time, Time-interval* are the designations of thematic roles (or conceptual cases); ex1 = "(Yves) has changed 700 franks", ex2 = "(Yves) has changed (700 franks) in the exchange office No. 14", ex3 = "(Yves) has changed (700 franks in the exchange office No. 14) on the 4th of March", ex4 = "Mary has changed (very much since last summer)"; ex5 = "(Mary) has changed (very much) since last summer"; ex6 = "The town has changed very much during the 2000s)". The fragments outside the parentheses are just the fragments where the considered thematic role (in other terms, a conceptual case) is realized. The fragments inside the parentheses only complement the fragments of the first kind in order to form a sentence.

Matrix Semantic-Syntactic Representations of NL-texts

Following (Fomichov 2010), let's consider a new data structure called *a matrix semantic-syntactic representation (MSSR)* of a natural language input text T. This data structure will be used for representing the intermediate results of semantic-syntactic analysis on a NL-text. A MSSR of a NL-text T is a string-numerical matrix *Matr* with the indices of columns or the groups of columns

locunit, nval, prep, posdir, reldir, mark, qt, nattr ,

it is used for discovering the conceptual (or semantic) relations between the meanings of the fragments of the text T, proceeding from the information about linguistically correct short word combinations. Besides, a MSSR of a NL-text allows for selecting one among several possible meanings of an elementary lexical unit. The number of the rows of the matrix *Matr* equals to *nt* - the number of the rows in the classifying representation *Rc*, i.e. it equals to the number of elementary meaningful text units in T.

Let's suppose that *k* is the number of arbitrary row from MSSR *Matr*. Then the element *Matr[k, locunit],* i.e. the element on the intersection of the row k and the column with the index *locunit* is the least number of a row from the array *Arls* (it is the projection of the lexico-semantic dictionary *Lsdic* on the input text T) corresponding to the elementary meaningful lexical unit *Rc[k, unit]*. It is possible to say that the value *Matr[k, locunit]* for the *k*-th elementary meaningful lexical unit from T is the coordinate of the entry into the array *Arls* corresponding to this lexical unit .

The column *nval* of *Matr* is used as follows. If *k* is the ordered number of arbitrary row in *Rc* and *Matr* corresponding to the elementary meaningful lexical unit, then the initial value of *Matr[k, nval]* is equal to the quantity of all rows from *Arls* corresponding to this lexical unit; that is, corresponding to different meanings of this lexical unit. When the construction of *Matr* is finished, the situation is to be different for all lexical units with several possible meanings: for each row of *Matr* with the ordered number *k* corresponding to a lexical unit, *Matr[k, nval]* = 1. because a certain meaning was selected for each elementary meaningful lexical unit.

For each row of *Matr* with the ordered number *k* associated with a noun or an adjective, the element in the column *prep* (preposition) specifies the preposition (possibly, the void, or empty, preposition *nil*) relating to the lexical unit corresponding to the k-th row.

Let's consider the purpose of introducing the column group

$$posdir\ (posdir_1,\ posdir_2,\ \ldots,\ posdir_n),$$

where n is a constant between 1 and 10 depending on the sprogram implementation. Let $1 \le d \le n$. Then we will use the designation $Matr[k,\ posdir,\ d]$ for an element located at the intersection of the k-th row and the d-th column in the group $posdir$. If $1 \le k \le nt,\ 1 \le d \le n,$ then $Matr[k,\ posdir,\ d] = m$, where m is either 0 or the ordered number of the d-th lexical unit wd from the input text T, where wd governs the text unit with the ordered number k.

There are no governing lexical units for the verbs in the principal clauses of the sentences, that is why for the row with the ordered number m associated with a verb, $Matr[m,\ posdir,\ d] = 0$ for any d from 1 to n. Let's agree that the nouns govern the adjectives as well as govern the designations of the numbers (e.g. "5 scientific articles"), cardinal numerals, and ordinal numerals. The group of the columns *reldir* consists of semantic relations whose existence is reflected in the columns of the group *posdir*. For filling in these columns, the templates (or frames) from the arrays *Arls, Arvfr, Arfrp* are to be used; the method can be grasped from the analysis of the algorithm *BuildMatr1* constructing a matrix semantic-syntactic representation of an input NL-text stated in (Fomichov 2010).

The column with the index *mark* is to be used for storing the variables denoting the different entities mentioned in the input text (including the events indicated by verbs, participles, gerunds, verbal nouns). The column *qt* (quantity) equals either to 0 or to the designation of the number situated in the text before a noun and connected to a noun. The column *nattr* (number of attributes) equals either to 0 or to the quantity of adjectives related to a noun presented by the k-th row, if we suppose that $Rc[k,\ unit]$ is a noun.

According to the method introduced in Chapter 8 of (Fomichov 2010), a MSSR of a NL-text T is used as an intermediary data structure for constructing a semantic representation of T being an expression of a certain SK-language (that is, being a K-representation of T). This transformation is performed by the algorithm of semantic assembly *BuildSem1* described in Chapter 10 of (Fomichov 2010).

Example. Let T1 be the question "Has the management board of the firm “Rainbow” changed in May?", and T1germ be the same question in German “Hat der Verwaltungsrat der Firma “Rainbow” in Mai veraendernt sich?”. Then it is possible to associate both with T1 and with T1germ the same K-representation *Semrepr1* of the form

*Question (x1, (x1 ≡ Truth-value(Situation(e1, change2 * (Focus-object,*
*certn manag-board * (Assoc-company, certn company1 * (Name1, “Rainbow”) : x3) : x2)*
(Time, Last-month(May, current-year)))))) .

Key Ideas of a Multilingual Algorithm Discovering Semantic Connections of the Verbs

Let us consider the conditions required for the existence of a semantic relationship between a meaning of a verbal form and a meaning of a word or word combination depending in a sentence on this verbal form. Let's agree to use the term "noun group" for designating the nouns or the

nouns together with the dependent words representing the concepts, objects and sets of objects. For example, let S1 = "When and where two aluminum containers with ceramic tiles have been delivered from?", S2 = "When the article by professor P. Somov was delivered?" and S3 = "Put the blue box on the green case". Then the phrases "two aluminum containers", "the article by professor P. Somov", "blue box" are the noun groups.

Let's call "a verbal form" either a verb in personal or infinitive form or a participle or a gerund. A discovery of possible semantic relationships between a verbal form and a phrase including a noun or an interrogative pronoun is playing an important role in the process of semantic-syntactic analysis of NL-texts.

Let's suppose that posvb is the position of a verbal form in the representation *Rc, posdepword* is the position of a noun or an interrogative pronoun in the representation *Rc*. The input data of the algorithm "Find-set-relations-verb-noun" are the integers *posvb, posdepword*, and two-dimensional arrays *Arls, Arvfr*, where *Arls* is the projection of the lexico-semantic dictionary *Lsdic* on the input text, *Arvfr* is the projection of the dictionary of verbal-prepositional frames *Vfr* on the input text.

The purpose of the algorithm "Find-set-relations-verb-noun" is in the first place to find the integer number *nrelvbdep* - the quantity of possible semantic relationships between the values of the text units with the numbers *p1* and *p2* in the classifying representation *Rc*. Secondly, this algorithm should build an auxiliary two-dimensional array *Arrelvbdep* keeping the information about possible semantic connections between the units of *Rc* with the numbers *p1* and *p2*. The rows of this array represent the information about the combinations of a meaning of the verbal form and a meaning of the dependent group of words (or one word).

The structure of each row of the two-dimensional array *Arrelvbdep* with the indices of columns *linenoun, linevb, role, example* is as follows. For the filled in row with the number *k* of the array *Arrelvbdep (k ≥1), linenoun* is the ordered number of the row of the array *Arls* corresponding to the word in the position *p1*; *linevb* is the ordered number of the row of the array *Arls* corresponding to the verbal form in the position *p2*; *role* is the designation of the semantic relationship (thematic role) connecting the verbal form in the position *p2* with the dependent word in the position *p1*; *example* is an example of an expression in NL realizing the same thematic role.

The search of the possible semantic relationships between a meaning of the verbal form (VF) and a meaning of the dependent group of words (DGW) is done with the help of the projection of the dictionary of verbal-prepositional frames (d.v.p.f.) Arvfr on the input text. In this dictionary such a frame (or a template) is searched that it would be compatible with the certain semantic-syntactic characteristics of the VF in the position posvb and the DGW with the number posdepword in Rc. Such characteristics include, first of all, the set of codes of grammatic cases *Grcases* associated with the text-forming unit having the ordered number - value *posdepwd* ("the position of dependent word") in *Rc*. Let's suppose that *Rc [posvb, tclass]=verb*. Then *Grcases* is the set of grammatic cases corresponding to the noun in the position *posdepword*.

Description of an Algorithm Discovering Semantic Relations Between a Verb and a Noun Group

Purpose of the Algorithm "Find-set-relations-verb-noun"

The algorithm is to establish a thematic role connecting a verbal form in the position *posvb* with a word (noun or connective word) in the position *posdepword* taking into account a possible preposition before this word. As a consequence, to select one of the several possible values of a verbal form and one of the several possible values of a word in the position *posdepword*. In order to do this, three enclosed loops are required: (1) with the parameter corresponding to a possible meaning of the word in the position *posdepword*; (2) with the parameter corresponding to a possible meaning of the verbal form; (3) with the parameter corresponding to a verbal-prepositional frame associated with this verbal form.

External specification of the algorithm "Find-set-relations-verb-noun"

Input: *input-lang* – string with the values *eng, germ, rus* denoting the English, German, and Russian languages; *Rc* - classifying representation, *nt* - integer - quantity of the text units in the classifying representation *Rc*, i.e. the quantity of rows in *Rc, Rm* - morphological representation of the lexical units from *Rc, posvb* - integer - position of a verbal form (a verb in a personal or infinitive form, or a participle or a gerund), *posdepword* - integer - position of a noun, *Matr* - initial value of MSSR of the text; *Arls* - array - projection of the lexico-semantic dictionary *Lsdic* on the input text T; *Arvfr* - array - projection of the dictionary of verbal-prepositional frames *Vfr* on the input text T.

Output: *arrelvbdep* - one-dimensional array designed to represent the information about (a) a meaning of a dependent word, (b) a meaning of a verbal form, and (c) about a semantic relationship between the verbal form in the position *posvb* and the dependent word in the position *posdepword; nrelvbdep* - integer – the quantity of meaningful rows in the array *arrelvbdep*.

External specification of the auxiliary algorithm "Characteristics-of-verbal-form"

Input: p1 - the number of a row from the classifying representation *Rc* corresponding to a verb or a participle.

Output: *form1, refl1, voice1* - strings; their values are defined in the following way. If p1 is the position of a verb, then form1 may have one of the following values: *indic* (the sign of the indicative mood), infinit (the sign of the infinitive form of a verb), *imperat* (the sign of the imperative mood). If p1 is the position of a participle, then form1 := *indic*. The string *refl1* takes the values *rf* (reflexive verb) or *nrf* (non-reflexive verb). The string *voice1* takes the value *actv* (the sign of the active voice) or *passv* (the sign of the passive voice). The values of the parameters *form1, refl1, voice1* are calculated based on the set of the numeric codes of the values of the morphological characteristics of the text unit with the ordered number *p1*.

External specification of the auxiliary algorithm "Range-of-sort"

Input: z - sort, i.e. an element of the set *St (B (Cb (Lingb)))*, where *Lingb* is a linguistic basis (see Chapter 7 of (Fomichov 2010)), *Cb (Lingb)* is a marked-up conceptual basis, *B (Cb (Lingb))* is the conceptual basis being the first component of *Cb (Lingb)*, *St (B (Cb (Lingb)))* is the set of sorts determined by the conceptual basis *B (Cb (Lingb))*.

Output: *spectrum* - set of all sorts being the generalizations of the sort *z* , including the sort *z* itself.

Algorithm "Find-set-relations-verb-noun"

```
Begin   Characteristics-of-verbal-form (posvb, form1, refl1, voice1)
        nrelvbdep := 0
Comment
Now the preposition is being defined
End-of-comment
        prep := leftprep
Comment
Calculation of  posn1  - position of the noun that defines the set of sorts of the text unit in the
position  posdepword
End-of-comment
        posn1 := posdepword
Comment
Then the set of grammatic cases  Grcases  is being formed. This set will be connected with the
word in the position  posdepword  in order to find a set of semantic relationships between the
words in the positions  posvb  and  posdepword.
End-of-comment
        t1 := Rc [posvb, tclass]
        t2 := Rc [posvb, subclass]
        p1 := Rc [posdepword, mcoord]
        Grcases := Cases (Rm [p1, morph])
        line1 := Matr [posn1, locunit]
        numb1 := Matr [posn1, nval]
Comment
The quantity of the rows with the noun meanings in  Arls
End-of-comment
        loop  for  i1  from  line1  to  line1 + numb1 - 1
Comment
A loop with the parameter being the ordered number of the row of the array  Arls  corresponding
to the noun in the position  posn1
End-of-comment
        Set1 := empty  set
        loop  for  j  from  1  to  m
Comment
 m  - semantic dimension of the sort system  S(B(Cb (Lingb))),  i.e. the maximal quantity of
incomparable sorts that may characterize one entity
```

```
End-of-comment
        current-sort := Arls [i1, st_j]
        if  current-sort ≠ nil
        then  Range-of-sort (current-sort, spectrum)
              Set1 := the union of the set Set1 and the set spectrum
        end-if
Comment
For an arbitrary sort  z  the value  spectrum  is the set of all sorts being the generalizations of the
sort  z  including the sort  z  itself
End-of-comment
        end-of-loop
Comment
End of the loop with the parameter  j
End-of-comment
Comment
Then the loop with the parameter corresponding to a meaning of the verbal form follows
End-of-comment
        line2 := Matr [posvb, locunit]
        numb2 := Matr [posvb, nval]
Comment
The quantity of the rows with the meanings of the verbal form in  Arls
End-of-comment
        loop  for  i2  from  line2  to  line2 + numb2 - 1
Comment
A loop with the parameter being the ordered number of a row of the array  Arls  corresponding to
the verb in the position  posvb
End-of-comment
        current-pred := Arls [i2, sem]
        loop  for  k1  from  1  to  narvfr
        if  Arvfr [k1, semsit] = current-pred
        then  begin   s1 := Arvfr [k1, str]
                      if  ((input-lang = Arvfr[k1, lang] and (prep = Arvfr [k1, sprep])
                           and   (s1 belongs to Set1)  and  (form1 = Arvfr [k1, fm])
                           and  (refl1 = Arvfr [k1,  refl])  and  (voice1 = Arvfr [k1, vc]))
                      then  grc := arvfr [k1, grcase]
                            if  (grc belongs to Grcases)
                            then
Comment
The relationship exists
End-of-comment
                 nrelvbdep := nrelvbdep + 1
               arrelvbdep [nrelvbdep, linevb] := i2
             arrelvbdep [nrelvbdep, linenoun] := i1
           arrelvbdep [nrelvbdep, gr] := grc
      arrelvbdep [nrelvbdep + 1, role] := arvfr [k1, trole]
                            end-if
```

```
                    end-if
                end
            end-if
            end-of-loop
            end-of-loop
            end-of-loop
Comment
End of loops with the parameters  i1,  i2,  k1
End-of-comment
 end
```

Commentary on the Algorithm "Find-set-relations-verb-noun"

The quantity *nrelvbdep* of the semantic relationships between the verbal form and a noun depending on it in the considered sentence is found. Let's consider such sublanguages of English, German, and Russian languages that in all input texts a verb is always followed (at certain distance) by at least one noun.

The information about such combinations of the meanings of the verb V and the noun N1 that give at least one semantic relationship between V and N1 is represented in the auxiliary array *arrelvbdep* with the indices of the columns *linenoun, linevb, role, example*. For arbitrary row of the array *arrelvbdep*, the column *linenoun* contains c1 - the number of such row of the array *Arls* that *Arls [c1, ord] = posn1* (position of the noun N1). For example, for Q1 = "When and where 3 aluminum containers have been delivered from?" *Arls [c1, sem] = container1*.

The column *linevb* contains c2 - the number of a row of the array *Arls* for which *Arls [c2, ord] = posvb*, i.e. the row c2 indicates a certain meaning of the verb V in the position *posvb*. For example, for Q1 = "When and where 3 aluminum containers have been delivered from?" the column *Arls [c2, sem] = delivery2*.

The column *role* is designed to represent the possible semantic relationships between the verb V and the noun N1. If *nrelvbdep* = 0 then the semantic relationships have not been found. Let's assume that this is not possible for the considered input language. If *nrelvbdep* = 1 then the following meanings have been clearly defined: the meaning of the noun N1 (by the row c1), the meaning of the verb V (by the row c2), and the semantic relationship *arrelvbdep [nrelvbdep, role]*. For example, for the question Q1 the following relationships are true: V = "delivered", N1 = "containers", *nrelvbdep = 1, arrelvbdep [nrelvbdep, role] = Object1*. If *nrelvbdep > 1* then it is required to apply the procedure that addresses clarifying questions to the user and to form these questions based on the examples from the column example.

Example. Let T2 be the sentence in German "Dr. Kurt Stein hat in Mai den Verwaltungsrat der Firma ”Rainbow” eingetreten” (an English version of T2 is the sentence T2eng = “"Dr. Kurt Stein joined in Mai the management board of the firm “Rainbow”). The German verb “eintreten” has, in particular, the following meanings: (1) to stand up for, (2) to make comfortable shoes, (3)

to join an organization. The conceptual analysis of T2 will enable a hypothetical applied intelligent system to positively answer the question T1 = "Has the management board of the firm “Rainbow” changed in May?".

Let’s show some details of analyzing T2. Suppose that the projection of the dictionary of verbal-prepositional semantic-syntactic frames *Vfr* on the input text T1 may include the following subarray *Arvfr2fragm* of the array *Arvfr2*:

nb	semsit	lang	fm	refl	vc	trole	sprep	grc	str	expl
4	standing-up-for	germ	indic	nrf	actv	Supported-person	fuer	4	ints	expl1
4	making-comfortable	germ	indic	nrf	actv	Object-to-wear	nil	4	shoes	expl2
4	jsoining2	germ	indic	nrf	actv	New-org	nil	4	org	expl3

The number 4 in the column *nb* indicates the 4th position of the text unit “hat eingetreten” in the classifying representation of T2. The elements *germ, indic, nrf, actv* are interpreted as the values *German, indicative-mood, non-reflexive, active-voice* of the properties *language, form-of-verb, reflexivity, voice*; the elements *Supported-person, Object-to-wear, New-org* are the designations of thematic roles (or conceptual cases). The number 4 in the column *grc* designates the article Akkusativ in the German language. The elements *ints, shoes, org* are interpreted as the sorts *intelligent-system, shoes, organization.* The examples *expl1, expl2, expl3* are defined by the relationships

expl1 = “Paul hat fuer seinen Freund Jens eingetreten” (“Paul has stood up for his friend Jens”);
expl2 = “Jean hat seine Schuhe eingetreten” (“Jean has made comfortable his shoes”);
expl3 = “Helene hat die Firma IBM in Maerz eingetreten” (“Helene joined in March the company IBM”).

Suppose that the algorithm “Find-set-relations-verb-noun” looks for possible semantic relations (thematic roles) in the combination

(hat eingetreten, den Verwaltungsrat) (1)

from the text T2. Then input-lang = germ, posvb = 4 (the position of the combination “hat eingetreten” in the classifying representation (CR) of T2), posn1 = 7 (the position of the combination “den Verwaltungsrat” in CR of T2).

The values of the parameter *i1* of the first loop of the algorithm correspond to different meanings of the noun group in the position 7. Different values of the parameter *i2* of another loop of the algorithm correspond to three meanings of the verb “eintreten” reflected in the considered subarray *Arvfr2fragm* of the array *Arvfr2*.

The frame represented by the first row of the subarray *Arvfr2fragm* doesn’t matches the combination (1) due to the lack in T2 of the German preposition “fuer” (“for” in English). The second frame determined by the subarray *Arvfr2fragm* doesn’t matches the combination (1) too,

since any reasonably designed lexico-semantic dictionary *Lsdic* doesn't allow for associating the sort "shoes" with the semantic unit *manag-board* (corresponding to the word combinations "a management board" and "ein Verwaltungsrat". But the third frame from the subarray *Arvfr2fragm* matches the combination (1), hence the semantic relation *New-org* will be discovered.

Step by step, the modified algorithm *SemSynt1b* (it includes the modified procedure "Find-set-relations-verb-noun") will build the following K-representation *Semrepr2* of T2:

*Situation (e2, joining2 * (Agent1, certn person * (First-name, "Kurt")*
(Surname, "Stein") : x1) (Time, Last-month(May, current-year))
*(New-org, certn org * (Isa, mang-board)(Company-part,*
*certn company1 * (Name1, "Rainbow") : x3) : x2)).*

Obviously, it is not difficult to include into the knowledge base of a hypothetical information retrieval system such fragments that this system would give a positive reply to the initial question T1 = "Has the management board of the firm "Rainbow" changed in May?", proceeding from the information conveyed by the K-representation *Semrepr2*.

Conclusions

The new method of developing the algorithms of semantic-syntactic analysis of NL-texts introduced in (Fomichov 2010) was modified and illustrated above. The method has a number of significant advantages in comparison with other known methods of developing the algorithms of the kind. *Firstly*, the explicitness and fullness of the description of the algorithm *SemSynt1* in (Fomichov 2010) is many times higher than it is typical for the scientific publications on this problem (see, e.g., the paper (Popescu et al. 2003)). *Secondly*, the method doesn't foresee the construction of a pure syntactic representation of the analyzed NL-text: it is oriented at discovering the semantic relations between the elementary meaningful units of a text.

Thirdly, the algorithm *SemSynt1* is multilingual in the following sense. This algorithm allows for using the same semantic-syntactic part of a linguistic database for English, German, and Russian languages. The algorithm *SemSynt1* contains the fragments meaning the calls of language-dependent auxiliary procedures. These procedures find and join several parts of a compound verbal form and join them into one elementary meaningful text unit, associate a preposition with a noun, etc. However, the discovery of possible semantic relations between the elementary meaningful text units is language-independent, and this promises economic advantages in case when the significant information may be obtained from the sources in several natural languages.

It seems that the algorithm *SemSynt1* in its modified form described above can be used as a basis for designing multilingual conceptual information retrieval systems of the computer intelligent systems with adaptive planning capabilities.

References

1. Fomichov, V.A. (2010); Semantics-Oriented Natural Language Processing: Mathematical Models and Algorithms; New York, Dordrecht, Heidelberg, London, Springer (354 pp.)
2. Popescu, A.-M., Etzioni, O., Kautz, H. (2003); Towards a Theory of Natural Language Interfaces to Databases. In: Proceedings of the 8th International Conference on Intelligent User Interfaces; Miami, FL (pp. 149-157)
3. Wilks, Y. and C. Brewster (2006); Natural Language Processing as a Foundation of the Semantic Web; Foundations and Trends in Web Science, Vol. 1, No. 3 - 4, now Publishers Inc. (129 pp)

Collaborative Agent Design Research Center - IIAS Technical Reports

InterSymp-09 Proceedings of the focus symposium on 'Knowledge Management Systems" (ed. J. Pohl); InterSymp-2009, 21st International Conference on Systems Research, Informatics, and Cybernetics, Baden-Baden, Germany, August 3-August 7, 2009.

InterSymp-08 Proceedings of the focus symposium on 'Intelligent Software Tools and Services" (ed. J. Pohl); InterSymp-2008, 20th International Conference on Systems Research, Informatics, and Cybernetics, Baden-Baden, Germany, July 24-July 30, 2008.

InterSymp-07 Proceedings of the focus symposium on 'Representation of Context in Software Data -> Information -> Knowledge' (ed. J. Pohl); InterSymp-2007, 19th International Conference on Systems Research, Informatics, and Cybernetics, Baden-Baden, Germany, July 30-Aug. 4, 2007.

InterSymp-06 Proceedings of the focus symposium on 'Advances in Intelligent Software Systems' (ed. J. Pohl); InterSymp-2006, 18th International Conference on Systems Research, Informatics, and Cybernetics, Baden-Baden, Germany, Aug. 7-Aug. 12, 2006.

InterSymp-04 Proceedings of the focus symposium on 'Intelligent Software Systems for the New Infostructure' (ed. J. Pohl); InterSymp-2004, 16th International Conference on Systems Research, Informatics, and Cybernetics, Baden-Baden, Germany, Jul. 29-Aug. 5, 2004.

InterSymp-03 Proceedings of the focus symposium on ' Collaborative Decision-Support Systems' (ed. J. Pohl); InterSymp-2003, 15th International Conference on Systems Research, Informatics, and Cybernetics, Baden-Baden, Germany, Jul. 28-Aug. 1, 2003.

InterSymp-02 Proceedings of the focus symposium on ' Collaborative Decision-Support Systems' (ed. J. Pohl); InterSymp-2002, 14th International Conference on Systems Research, Informatics, and Cybernetics, Baden-Baden, Germany, Jul. 29-Aug. 3, 2002.

InterSymp-01 Proceedings of the focus symposium on 'Advances in Computer-Based and Web-Based Collaborative Systems' (eds. J. Pohl and T. Fowler); InterSymp-2001, 13th International Conference on Systems Research, Informatics, and Cybernetics, Baden-Baden, Germany, Jul. 30-Aug. 4, 2001.

InterSymp-00 Proceedings of the focus symposium on 'Advances in Computer-Based and Web-Based Collaborative Systems' (eds. J. Pohl and T. Fowler); InterSymp-2000, 12th International Conference on Systems Research, Informatics, and Cybernetics, Baden-Baden, Germany, Jul. 31, Aug. 4, 2000.

InterSymp-99 Proceedings of the focus symposium on 'Computer-Based and Web-Based Collaborative Systems' (eds. J. Pohl and T. Fowler); InterSymp-99 International Conference on Systems Research, Informatics, and Cybernetics, Baden-Baden, Germany, Aug.2-6, 1999.

InterSymp-98 Proceedings of the focus symposium on 'Collaborative Decision- Support Systems for Design, Planning, and Execution' (ed. J. Pohl); InterSymp-98 International Conference on Systems Research, Informatics, and Cybernetics, Baden-Baden, Germany, Aug.17-21, 1998.

InterSymp-97 Proceedings of the focus symposium on 'Collaborative Design and Decision-Support Systems' (ed. J. Pohl); InterSymp-97 International Conference on Systems Research, Informatics and Cybernetics, Baden-Baden, Germany, Aug.18-23, 1997.

InterSymp-96 Proceedings of the focus symposium on 'Advances in Cooperative Environmental Design Systems' (ed. J. Pohl); InterSymp-96 International Conference on Systems Research, Informatics and Cybernetics, Baden-Baden, Germany, Aug.14-18, 1996.

InterSymp-95 Proceedings of the focus symposium on 'Advances in Cooperative Computer-Assisted Environmental Design Systems' (ed. J. Pohl); InterSymp-95 8th International Conference on Systems Research, Informatics and Cybernetics, Baden-Baden, Germany, Aug. 16-20, 1995.

InterSymp-94 Proceedings of the focus symposium on 'Advances in Computer-Based Building Design Systems' (ed. J. Pohl); InterSymp-94 7th International Conference on Systems Research, Informatics and Cybernetics, Baden-Baden, Germany, Aug.15-21, 1994.

InterSymp-93 Proceedings of the focus symposium on 'Advances in Computer-Assisted Building Design Systems' (ed. J. Pohl); InterSymp-93 4th International Symposium on Systems Research, Informatics and Cybernetics, Baden-Baden, Germany, Aug.2-5, 1993.

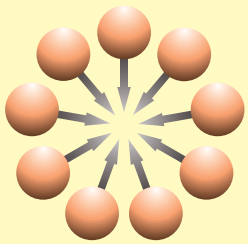
InterSymp-92 Proceedings of the focus symposium on 'Advances in Computer-Based Design Environments' (ed. J. Pohl); InterSymp-92 6th International Conference on Systems Research, Informatics and Cybernetics, Baden-Baden, Germany, Aug.18-19, 1992.

InterSymp-91 Proceedings of the focus symposium on 'Computer-User Partnerships in Design' (ed. J. Pohl); InterSymp-91 3rd International Symposium on Systems Research, Informatics and Cybernetics, Baden-Baden, Germany, Aug.13-14, 1991.

InterSymp-90 Proceedings of the focus symposium on 'Knowledge-Based Systems in Building Design' (ed. J. Pohl); InterSymp-90 5th International Conference on Systems Research, Informatics and Cybernetics, Baden-Baden, Germany, Aug.7-8, 1990.

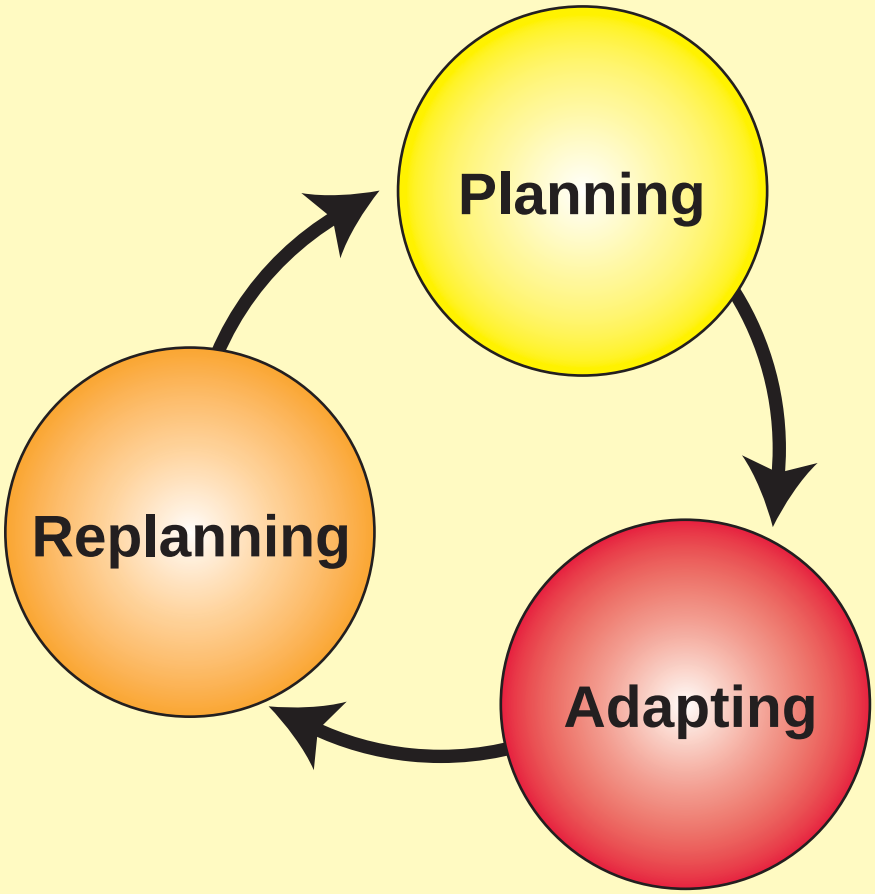
www.cadrc.calpoly.edu

ISBN 978-1-897233-17-7



CADRC

Collaborative Agent Design Research Center



Cal Poly
San Luis Obispo, CA 93407 USA

Phone: (805) 756-1310/2841
FAX: (805) 756-7568

e-mail: jpohl@calpoly.edu
www.cadrc.calpoly.edu

ISBN 978-1-897233-17-7