Intelligent Software for Ecological Building Design

Jens Pohl[1], Hisham Assal[1], and Kym Jason Pohl[2]

[1]Collaborative Agent Design Research Center (CADRC)
California Polytechnic State University (Cal Poly)
[2]Tapestry Solutions, 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.

Keywords

agents; architectural design; collaboration; design; ecological design; ontology; representation; service-oriented architecture (SOA); sustainability.

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 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[1]. 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?

[1] 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).

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 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 databases. Service-Oriented Architecture (SOA) concepts would appear to 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.

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 requesting 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 information it processes is the single most important

prerequisite for a collaborative design environment (Assal et al. 2009). An expressive representation of the real world objects 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 in its components. To the designer a building consists of real world objects, such as rooms, walls, windows, doors, structural components, furniture, and so on. Each of these objects has attributes and relationships that determine its behavior under certain conditions. These semantic descriptors form the basis of collaboration among human problem solvers and are therefore likewise the fundamental mode of communication 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 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 and discovery. 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 service requests and 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 focus on the identification

rather than the automatic resolution of conflicts. 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 that appear to have precipitated the conflict. The automatic tracing 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 due to the extensive set of relationships represented within the model that retains their connections. 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 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 behavior and results. 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 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 inference 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 sequence 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* 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 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 *reusable* 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)[2] 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.

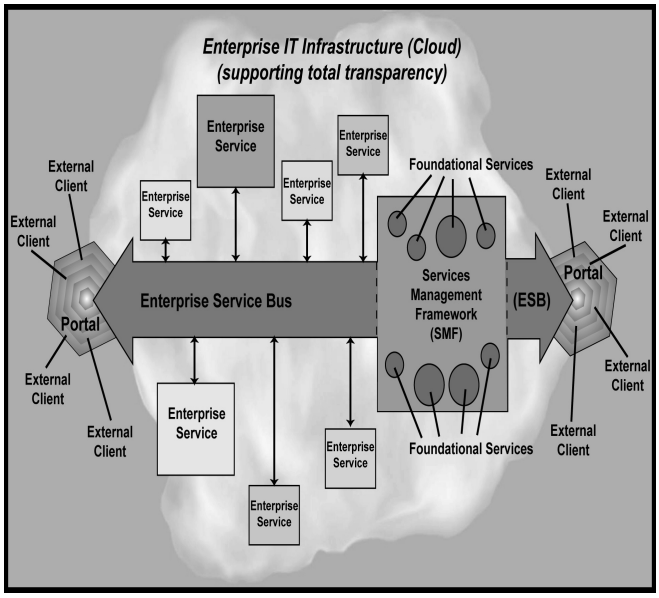

Figure 1: Principal SOA components

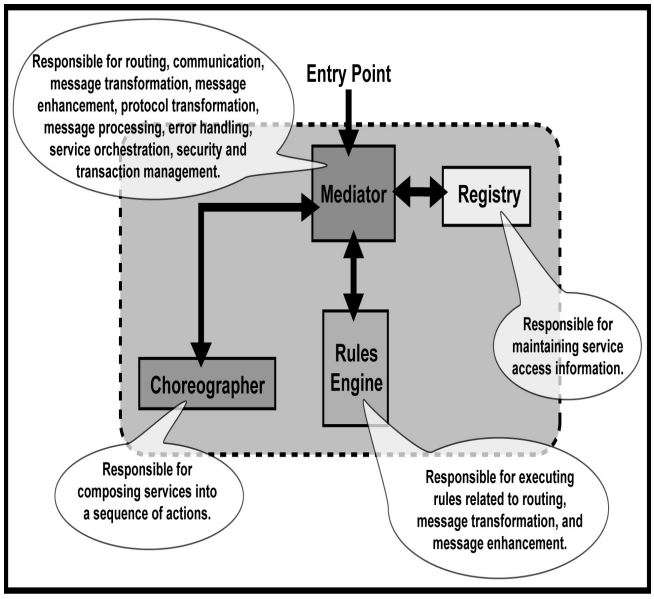

Figure 2: Principal ESB components

The principal components of a conceptual SOA implementation scheme (Figure 1) include a Services Management Framework (SMF), various kinds of foundational services that allow the SMF to perform its management functions, one or more portals to external clients, and the enterprise services that facilitate the ability of the user community to perform its operational tasks.

Services Management Framework (SMF): A Services Management Framework (SMF) is essentially a SOA-based software infrastructure that utilizes tools to manage the exchange of messages among enterprise services. The messages may contain requests for services, data, the results of services performed, or any combination of these. The tools are often referred to as foundational services because they are vital to the ability of the SMF to perform its management functions, even though they are largely hidden from the user community. The SMF must be capable of:

[2] 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.

- Undertaking any transformation, orchestration, coordination, and security actions necessary for the effective exchange of the message.
- Maintaining a loosely coupled environment in which neither the service requesters nor the service providers need to communicate directly with each other; - or even have knowledge of each other.

A SMF may accomplish some of its functions through an Enterprise Service Bus (ESB), or it may be implemented entirely as an ESB.

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 manages 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.

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, rich in relationships that 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). 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.

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, typically referred to as *services*. If these services include a version of the underlying ontology then they are able to communicate 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 software code of each client includes a version of the ontology only the changes in information need to be communicated. For

example, an agent that is monitoring the position of spaces during the design of the floor plan of a building may have more than 100 objects included in its subscription profile. One set of these objects will represent the location, geometric parameters and functional characteristics of a particular space. If the designer changes the locations of this space then in reality only one attribute (i.e., the location attribute) of one object may have changed. Only the changed value of this single object needs to be transmitted to the agent, since it already has all of the information that has not changed.

Second, each client can register a standing request for the kind of information that it would like to receive. This is referred to as a subscription profile, and the client has the ability to change this profile dynamically during execution if it sees cause to ask for additional or different information. By allowing information to be automatically *pushed* to clients, the subscription service obviates the need for database queries and thereby greatly reduces the amount of work the computer has to perform. Of course, a separate query service is also usually provided so that a client can make one-time requests for information that is not required on a continuous basis.

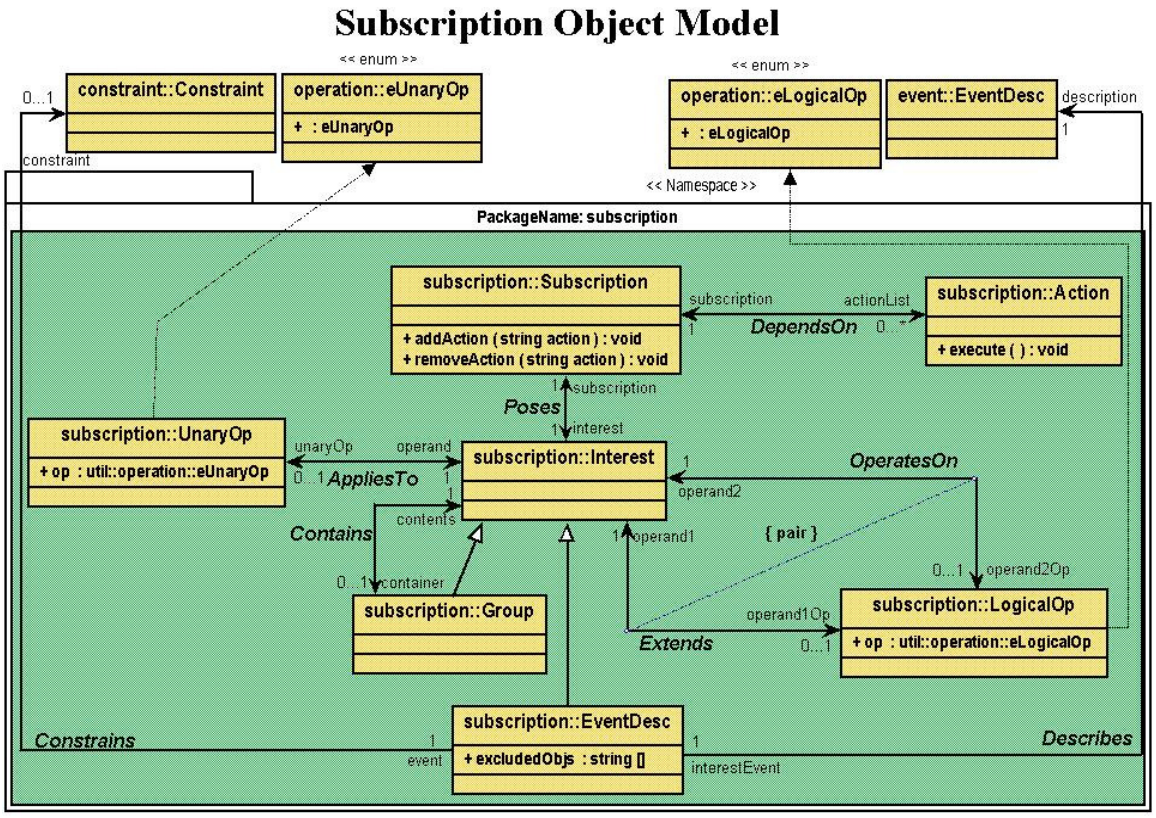

Figure 3: Typical subscription service domain ontology

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.

The ability of both the services and the human users to communicate at the higher level of information (i.e., data in context) within a SOA-based software system is the most fundamental requirement of an intelligent design environment. It is an essential prerequisite for the meaningful interaction of the human designer with the various design tools, as well as the ability of those services to reactively monitor the design process and proactively contribute in a meaningful manner to the evolving design solution.

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 services (i.e., design tools) that are available to the human designer. Most of these services will either consist of or include agents. In the broadest sense an agent may be described as a computer-based application that has communication capabilities to external entities and can perform some useful tasks in at least a semi-autonomous fashion.

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 objects and their relationships 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 many of the relationships and associations 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 interact 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 these may be appropriate. This requires agents to be 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 be automatically triggered 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 objects and the current requirements. This planning process involves matching the requirements with the available resources to produce a course of action that will deliver the resources to the requesting objects. The complexity of the process can be reduced by distributing the basic planning tasks among a set of agents, as follows: identify the requirements; identify the available resources; report any shortage of resources for a given set of requirements; identify the available set of actions; and, generate a plan for fulfilling the requirements.

Plan 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 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 narrow domain, and perform planning or assessment tasks for 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 application. Service Agents can be designed to respond to changes in the problem state spontaneously through their ability to monitor information changes and respond opportunistically, or information may be passed to them in some chronological order based on time-stamped events or predefined priorities. They should be able to generate queries dynamically and access data sources automatically whenever the need arises.

In the proposed intelligent design environment these agents will constitute the principal design tools by providing analysis, solution generation and evaluation capabilities for the full range of knowledge domains that impact a 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 agents will be able to operate in parallel in the proposed design environment. Furthermore, the ability of the agents to communicate 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 an agent type based on the agentification of the information objects that are intrinsic to the nature of each application. In the proposed design environment these are the information objects 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). 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 communication capabilities, objects can pursue their own needs and perform a great deal of local problem solving without continuously impacting the communication and coordination facilities utilized by the higher level components of the distributed system. Typically, a Mentor Agent is a process (i.e., program) or component of a process that includes several adjuncts that provide the agent 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 request services through their communication facilities 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 associations to other objects. 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 objects to agent status. Objects that are subservient to other objects 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, objects 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 *Space* objects as agents. Since Mentor Agents have communication capabilities the *Space* agent would be able to request assistance from other agents such as Service Agents. If the *Space* agent would like to know where it is located in respect to its nearest neighbors it could broadcast a request for specific information relating to its spatial context. A Service Agent whose domain encompasses the computation of spatial locations would receive the request, perform the necessary calculations, and transmit the results back to the *Space* agent. Two immediate advantages have accrued: only the most necessary computation has been performed; and, the results of the computation that form part of the fundamental description of the object can be held anywhere in the system (as long as they are available to any other authorized agent). Second, by distributing the requestors, the requestees, and the information that is generated as a result of the servicing of the requests, the communications involved with both the current request transactions and any future use of the 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 systems developed by the authors (ICADS 1991, AEDOT 1992) conflicts arose when Service 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* 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 overrule the Service Agent. The conference room, as a passive object, 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 object 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 *Space* agent could broadcast two specific requests for service: *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 *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 *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 better solution. The search may continue into subsequent stages of the design process, during which the conference room becomes part of *Floor* and *Building* Mentor Agents. 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 is contrary 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 *Space* agent is able to communicate its proposal and have the noise control issue reconsidered. It could notify, 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 *Space* agent may appeal directly to the designer. The designer has several alternative courses of actions open: 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 view and use the *Space* agent's proposal as the basis for the exploration of an alternative solution path.

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 ontology offers a direction for the rich representation of domain knowledge, and intelligent agents are 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).

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.

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. (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 Sofware Design'; Technical Report 7, The Institute for the Learning Sciences, Northwestern University.

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

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

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

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.